Parcours d'un graphe

ISN 2013

Exercices à rendre

Trois exercices sont à rendre.

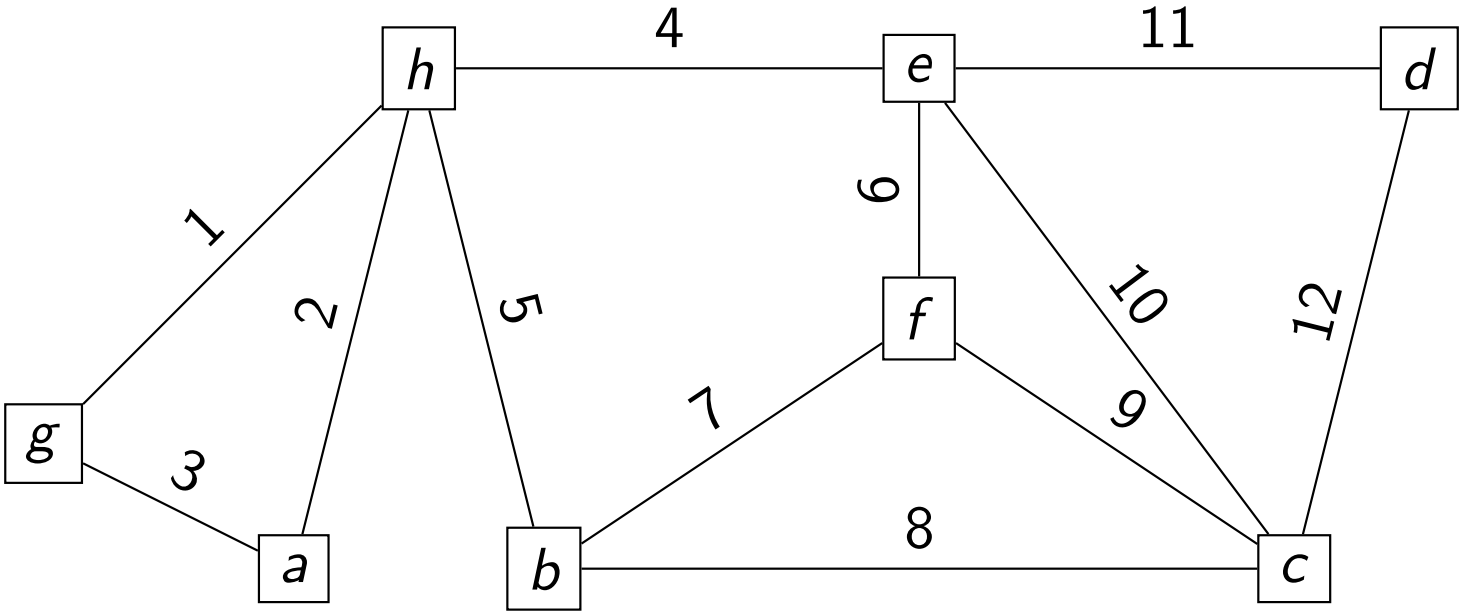
- L' exercice 1 pourra être rendu sur papier mardi 2 avril (ou en version électronique si vous préférez).
- Les exercices 2 et 3 sont à rendre dans les casiers numériques de vos enseignants lundi 1 avril.

A savoir

A la suite de cette séance, vous devrez **savoir** parcourir un graphe en profondeur et en largeur.

L'essentiel de la notion de graphe

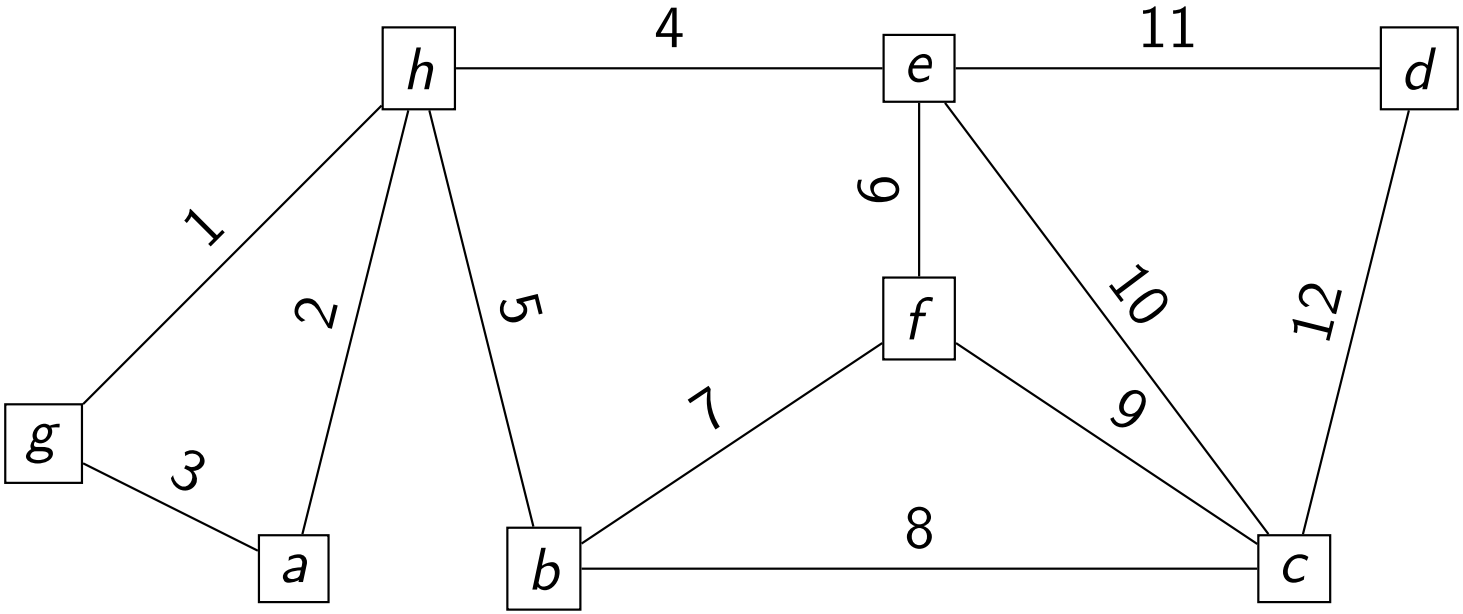
On peut appréhender la notion de graphe par l'une de ses représentations classiques : des points (sommets du graphe) et des courbes reliant certains de ces points (arêtes du graphe).



Les sommets de ce graphe sont a, b, c, d, e, f, g, h. Les sommets e et c sont adjacents (voisins) : ils sont en effet reliés par l'arête 10. Le sommet b a pour voisins h, f et c. Le sommet a est incident aux arêtes 2 et 3.

L'essentiel de la notion de graphe

On peut appréhender la notion de graphe par l'une de ses représentations classiques : des points (sommets du graphe) et des courbes reliant certains de ces points (arêtes du graphe).



Lorsqu'on passe d'un sommet à un autre en se déplaçant le long d'arêtes et de sommets, on dit que l'on définit un chemin dans le graphe. On peut par exemple définir le chemin g, 3, a, 2, h, 5, b, 7, f dans le graphe ci-dessus.

Exemples de situations modélisées par un graphe

Exemples de situations modélisées par un graphe

- Le web : chaque page est un sommet du graphe, chaque lien hypertexte est une arête entre deux sommets.

Exemples de situations modélisées par un graphe

- Le web : chaque page est un sommet du graphe, chaque lien hypertexte est une arête entre deux sommets.
- Un réseau ferroviaire : chaque gare est un sommet, les voies entre deux gares sont les arêtes. Idem avec un réseau routier.

Exemples de situations modélisées par un graphe

- Le web : chaque page est un sommet du graphe, chaque lien hypertexte est une arête entre deux sommets.
- Un réseau ferroviaire : chaque gare est un sommet, les voies entre deux gares sont les arêtes. Idem avec un réseau routier.
- Un réseau social : les sommets sont les personnes, deux personnes sont adjacentes dans ce graphe lorsqu'elles sont "amies".
  Si la notion d'amitié n'est pas réciproque, le graphe est orienté.

Exemples de situations modélisées par un graphe

- Le web : chaque page est un sommet du graphe, chaque lien hypertexte est une arête entre deux sommets.
- Un réseau ferroviaire : chaque gare est un sommet, les voies entre deux gares sont les arêtes. Idem avec un réseau routier.
- Un réseau social : les sommets sont les personnes, deux personnes sont adjacentes dans ce graphe lorsqu'elles sont "amies".
  Si la notion d'amitié n'est pas réciproque, le graphe est orienté.

La structure de graphe est en science de l'informatique une structure abstraite omniprésente.

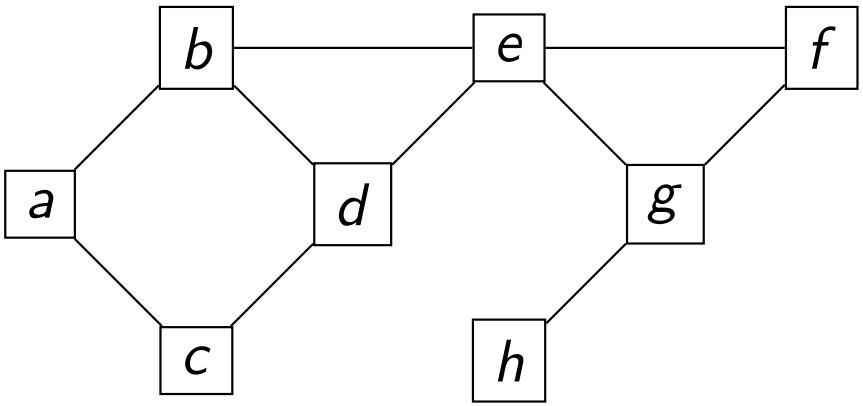
Représentation informatique d'un graphe

Une structure théorique comme un graphe est susceptible de nombreuses implémentations, selon le type de problèmes à résoudre.

Représentation informatique d'un graphe

Une structure théorique comme un graphe est susceptible de nombreuses implémentations, selon le type de problèmes à résoudre.

On peut par exemple utiliser la matrice d'adjacence du graphe.

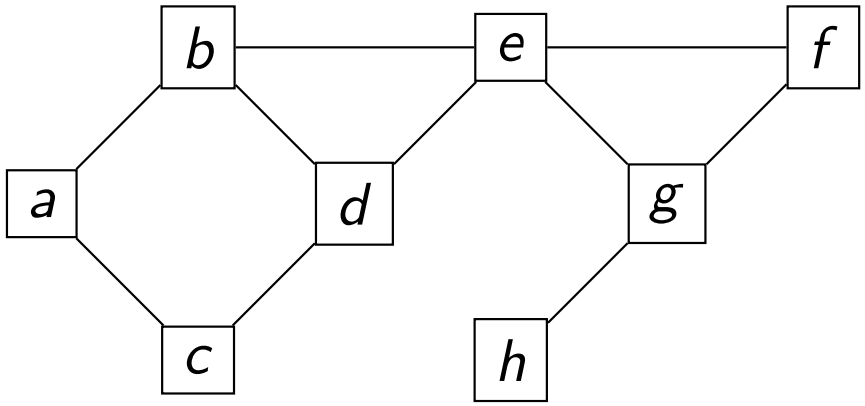


$$
\begin{array}{c} \\ a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \end{array}
\begin{array}{c}
\begin{array}{cccccccc} a & b & c & d & e & f & g & h \end{array} \\
\begin{pmatrix}
0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0
\end{pmatrix}
\end{array}
$$

Exemple de codage : utilisation d'un dictionnaire python

Python

```
G=dict()
G['a']=['b','c']
G['b']=['a','d','e']
G['c']=['a','d']
G['d']=['b','c','e']
G['e']=['b','d','f','g']
G['f']=['e','g']
G['g']=['e','f','h']
G['h']=['g']
```



PILE et FILE

Les notions de PILE et de FILE sont deux structures de données abstraites importantes en informatique.
On limite ci-dessous la présentation de ces notions aux besoins des parcours de graphes envisagés ci-après.

PILE (stack)

La structure de PILE est celle d'une pile d'assiettes :

- Pour ranger les assiettes, on les empile les unes sur les autres.
- Lorsqu'on veut utiliser une assiette, c'est l'assiette qui a été empilée en dernier qui est utilisée.

Structure LIFO (last in, first out)

FILE (queue)

La structure de FILE est celle d'une file d'attente à un guichet :

- Les nouvelles personnes qui arrivent se rangent à la fin de la file d'attente.
- La personne servie est celle qui est arrivée en premier dans la file.

Structure FIFO (first in, first out).

Parcours en largeur

Parcours en largeur : principe de l'algorithme

Vous devez parcourir toutes les pages d'un site web. Les pages sont les sommets d'un graphe et un lien entre deux pages est une arête entre ces deux sommets.

Parcours en largeur : principe de l'algorithme

Vous devez parcourir toutes les pages d'un site web. Les pages sont les sommets d'un graphe et un lien entre deux pages est une arête entre ces deux sommets.

1. Dans le parcours en largeur, on utilise une file. On enfile le sommet de départ (on visite la page index du site).

Parcours en largeur : principe de l'algorithme

Vous devez parcourir toutes les pages d'un site web. Les pages sont les sommets d'un graphe et un lien entre deux pages est une arête entre ces deux sommets.

1. Dans le parcours en largeur, on utilise une file. On enfile le sommet de départ (on visite la page index du site).
2. On visite les voisins de la tête de file (pages ciblées par la page de tête de file). On les enfile (en les numérotant au fur et à mesure de leur découverte) s'ils ne sont pas déjà présents dans la file, ni déjà passés dans la file.

Parcours en largeur : principe de l'algorithme

Vous devez parcourir toutes les pages d'un site web. Les pages sont les sommets d'un graphe et un lien entre deux pages est une arête entre ces deux sommets.

1. Dans le parcours en largeur, on utilise une file. On enfile le sommet de départ (on visite la page index du site).
2. On visite les voisins de la tête de file (pages ciblées par la page de tête de file). On les enfile (en les numérotant au fur et à mesure de leur découverte) s'ils ne sont pas déjà présents dans la file, ni déjà passés dans la file.
3. On défile (c'est à dire : on supprime la tête de file).

Parcours en largeur : principe de l'algorithme

Vous devez parcourir toutes les pages d'un site web. Les pages sont les sommets d'un graphe et un lien entre deux pages est une arête entre ces deux sommets.

1. Dans le parcours en largeur, on utilise une file. On enfile le sommet de départ (on visite la page index du site).
2. On visite les voisins de la tête de file (pages ciblées par la page de tête de file). On les enfile (en les numérotant au fur et à mesure de leur découverte) s'ils ne sont pas déjà présents dans la file, ni déjà passés dans la file.
3. On défile (c'est à dire : on supprime la tête de file).
4. On recommence au point 2 (tant que c'est possible, c'est à dire tant que la file n'est pas vide).

Parcours en largeur : principe de l'algorithme

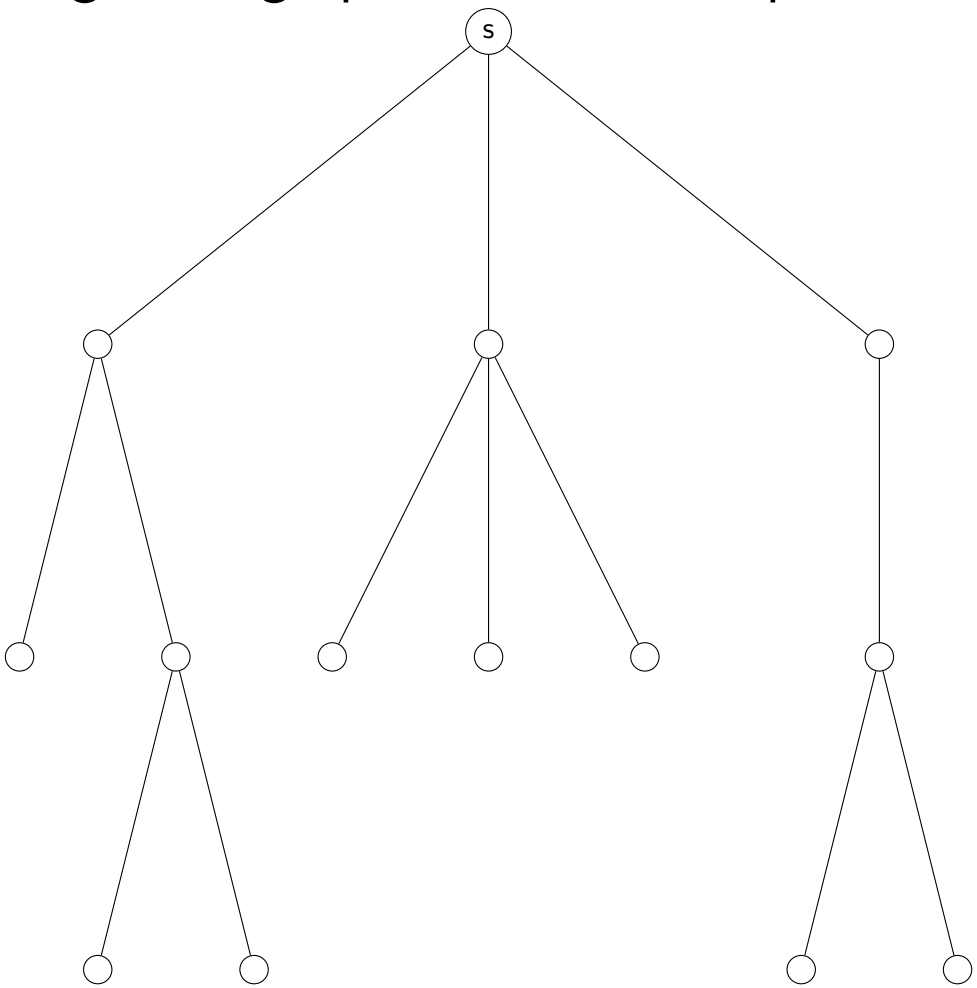
Vous devez parcourir toutes les pages d'un site web. Les pages sont les sommets d'un graphe et un lien entre deux pages est une arête entre ces deux sommets.

1. Dans le parcours en largeur, on utilise une file. On enfile le sommet de départ (on visite la page index du site).
2. On visite les voisins de la tête de file (pages ciblées par la page de tête de file). On les enfile (en les numérotant au fur et à mesure de leur découverte) s'ils ne sont pas déjà présents dans la file, ni déjà passés dans la file.
3. On défile (c'est à dire : on supprime la tête de file).
4. On recommence au point 2 (tant que c'est possible, c'est à dire tant que la file n'est pas vide).

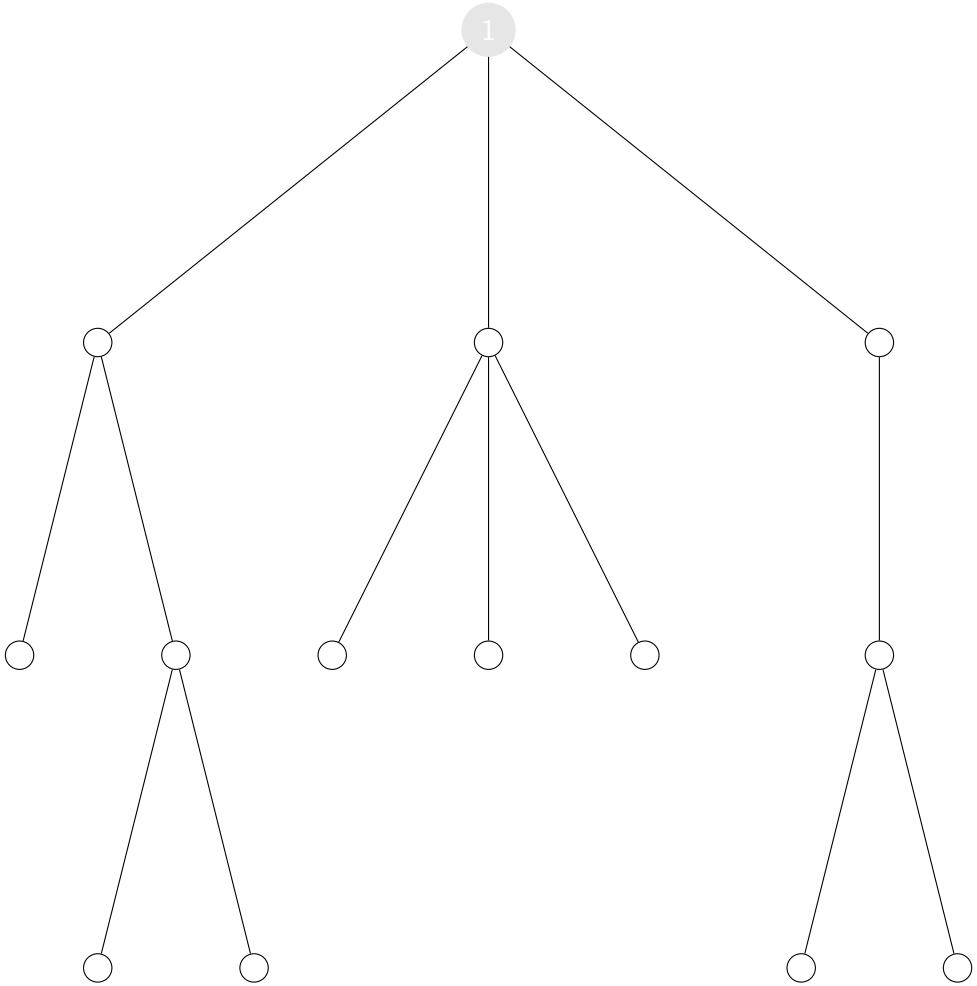
En d'autres termes, on traite toujours en priorité les liens des pages le plus tôt découvertes.

Parcours en largeur d'un arbre

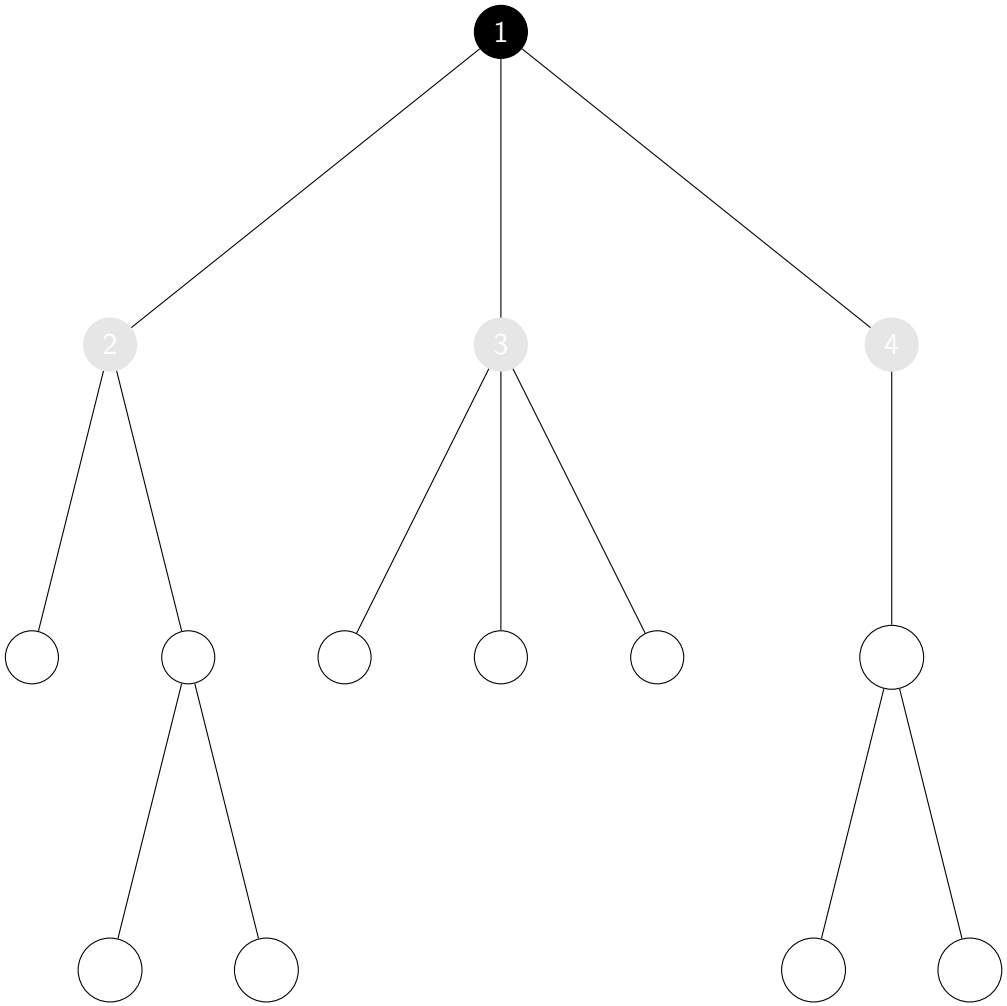
Parcourir en largeur le graphe ci-dessous à partir du sommet s :



Parcours en largeur d'un arbre



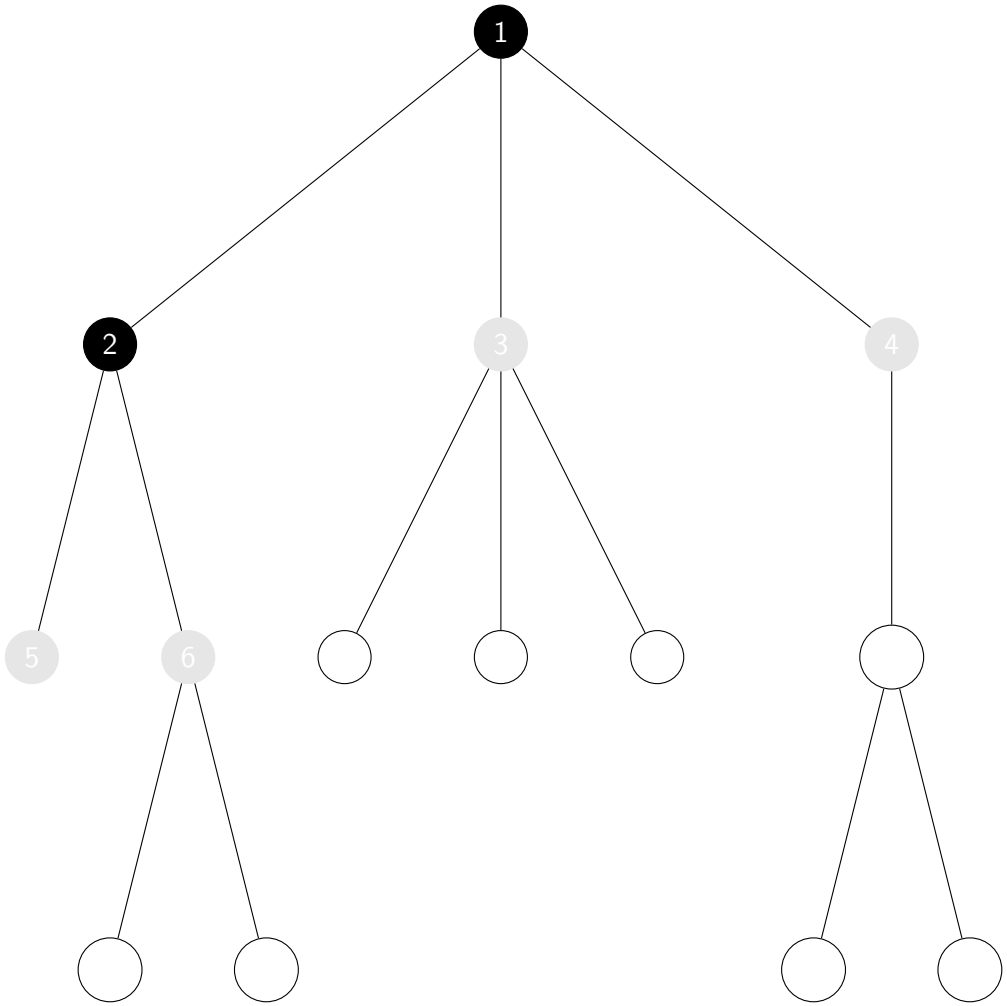
Parcours en largeur d'un arbre



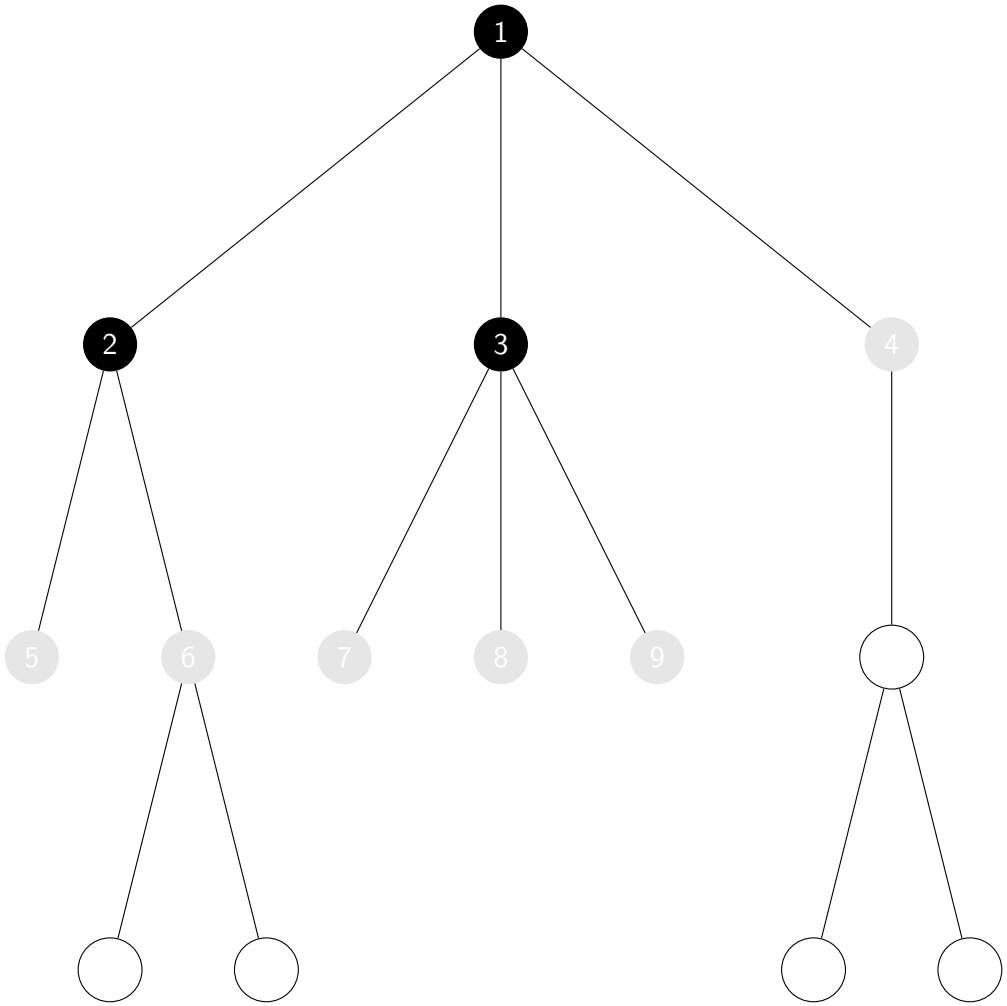
Enfiler : passage en gris. Défiler : passage en noir.

L'ordre pour enfiler les voisins (ni gris, ni noirs) dépend de l'implantation.

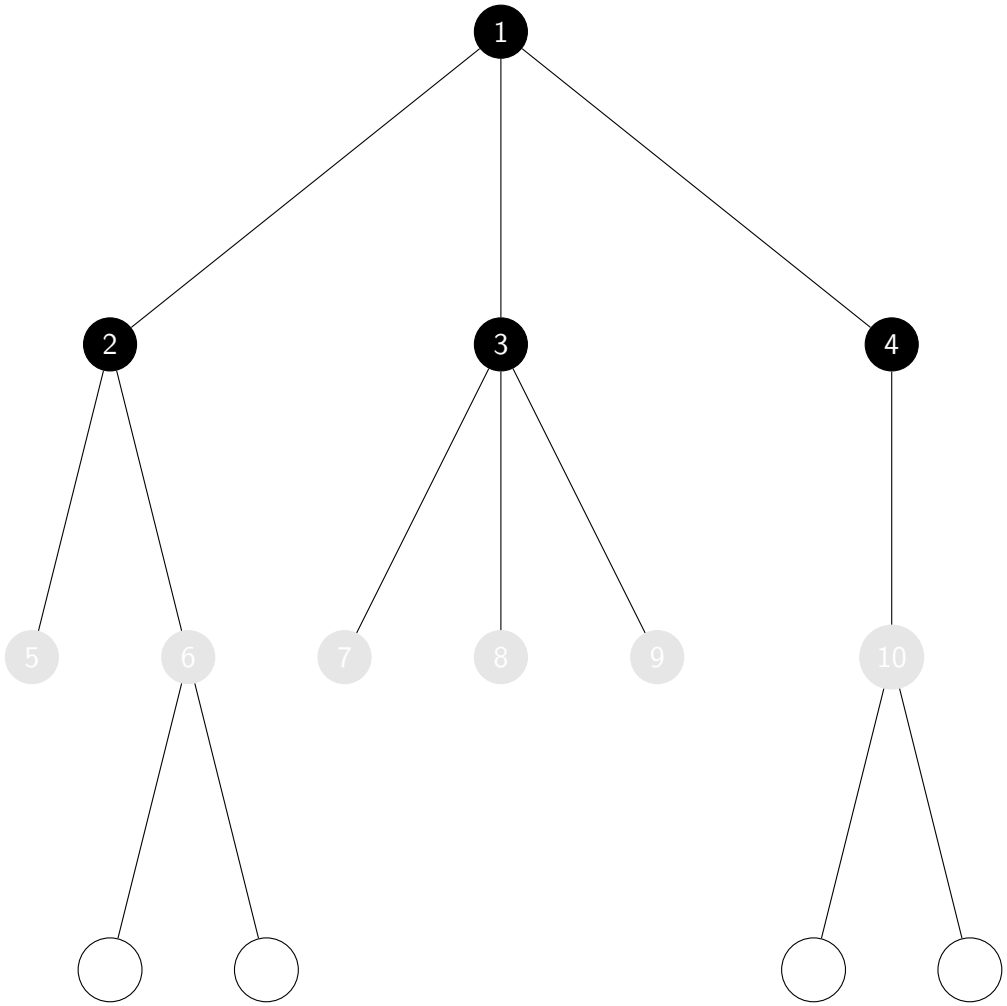
Parcours en largeur d'un arbre



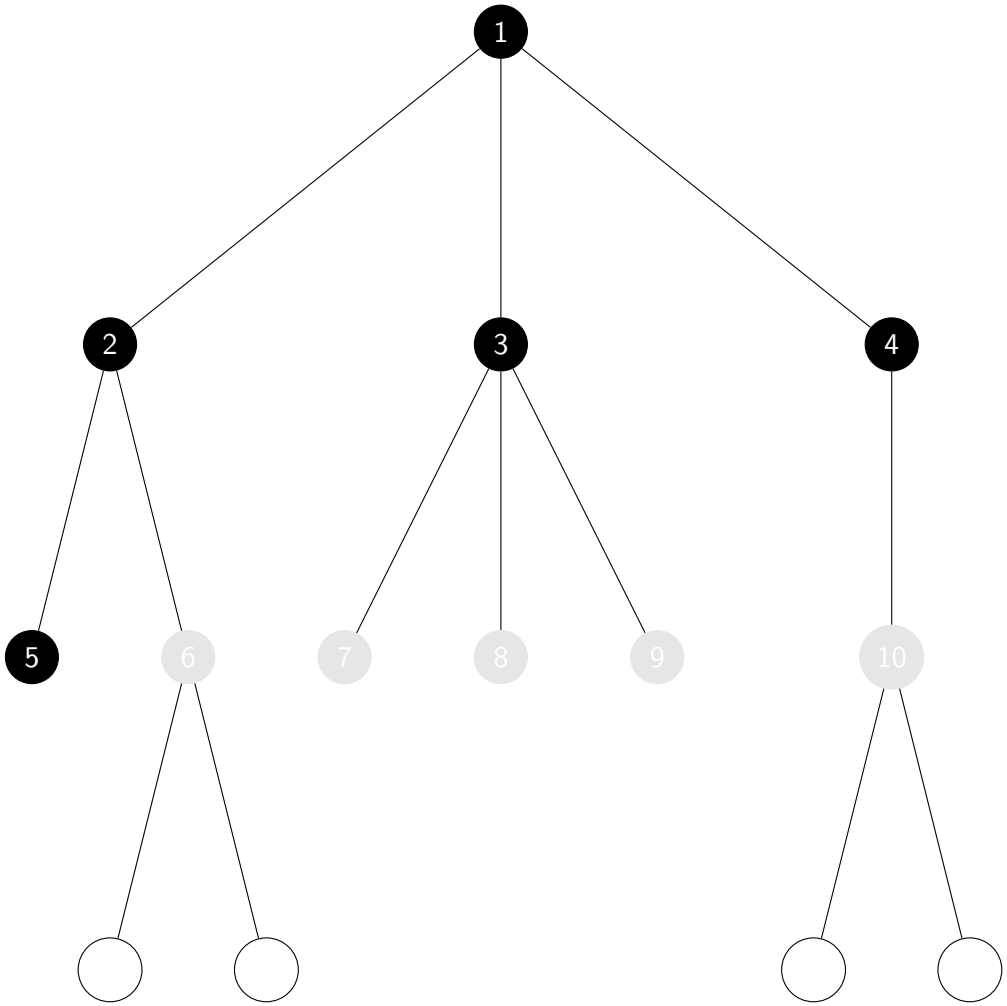
Parcours en largeur d'un arbre



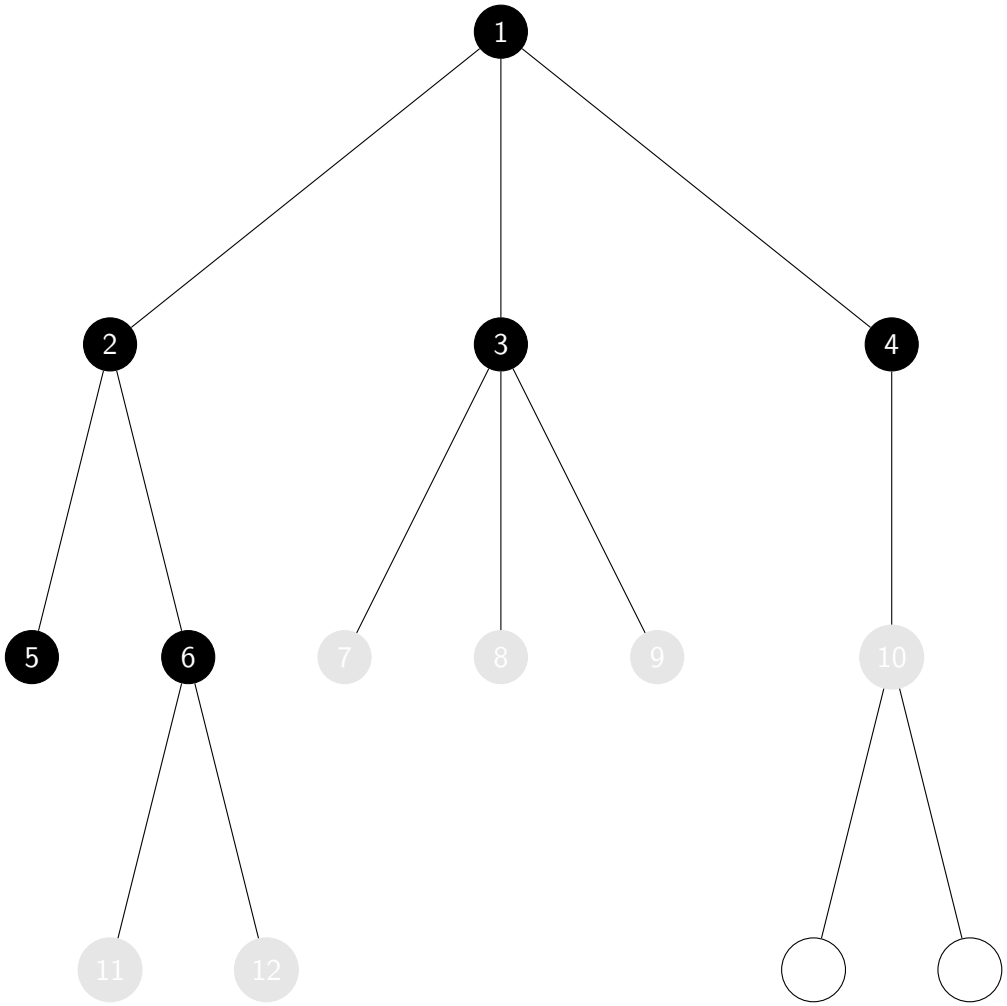
Parcours en largeur d'un arbre



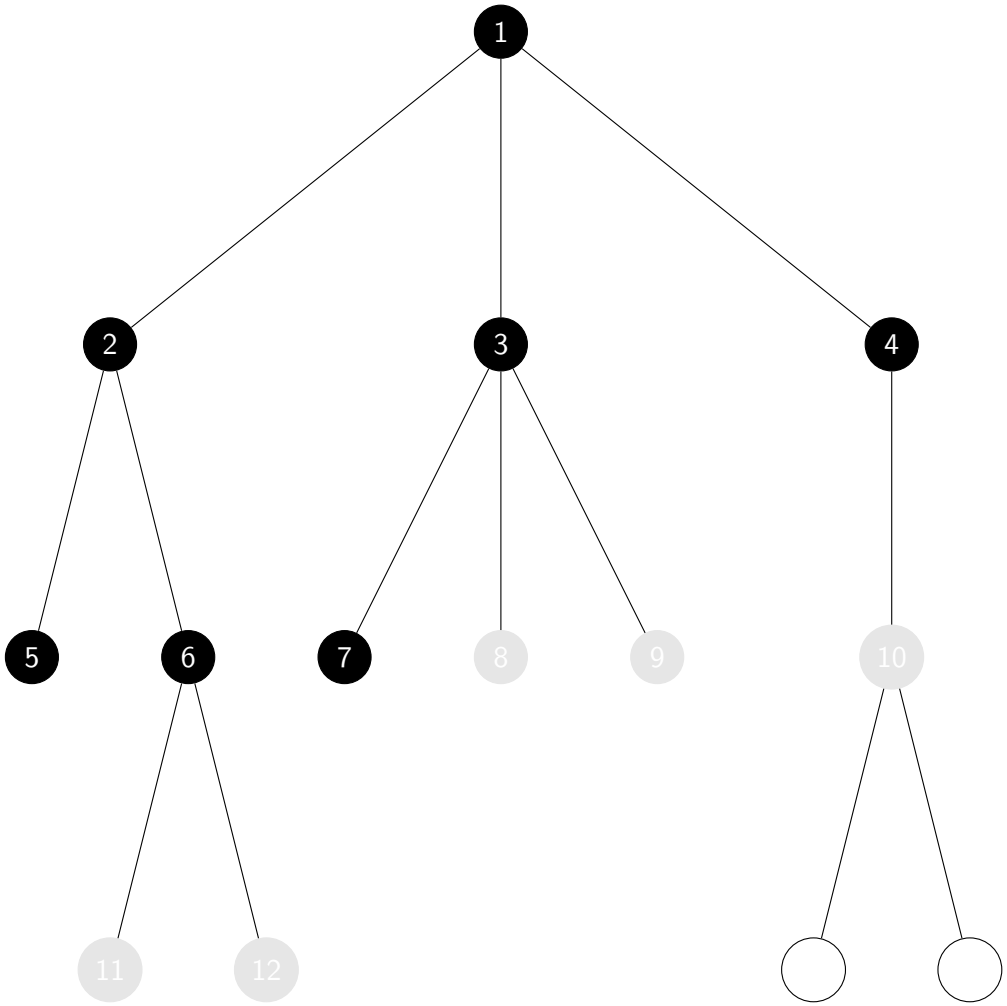
Parcours en largeur d'un arbre



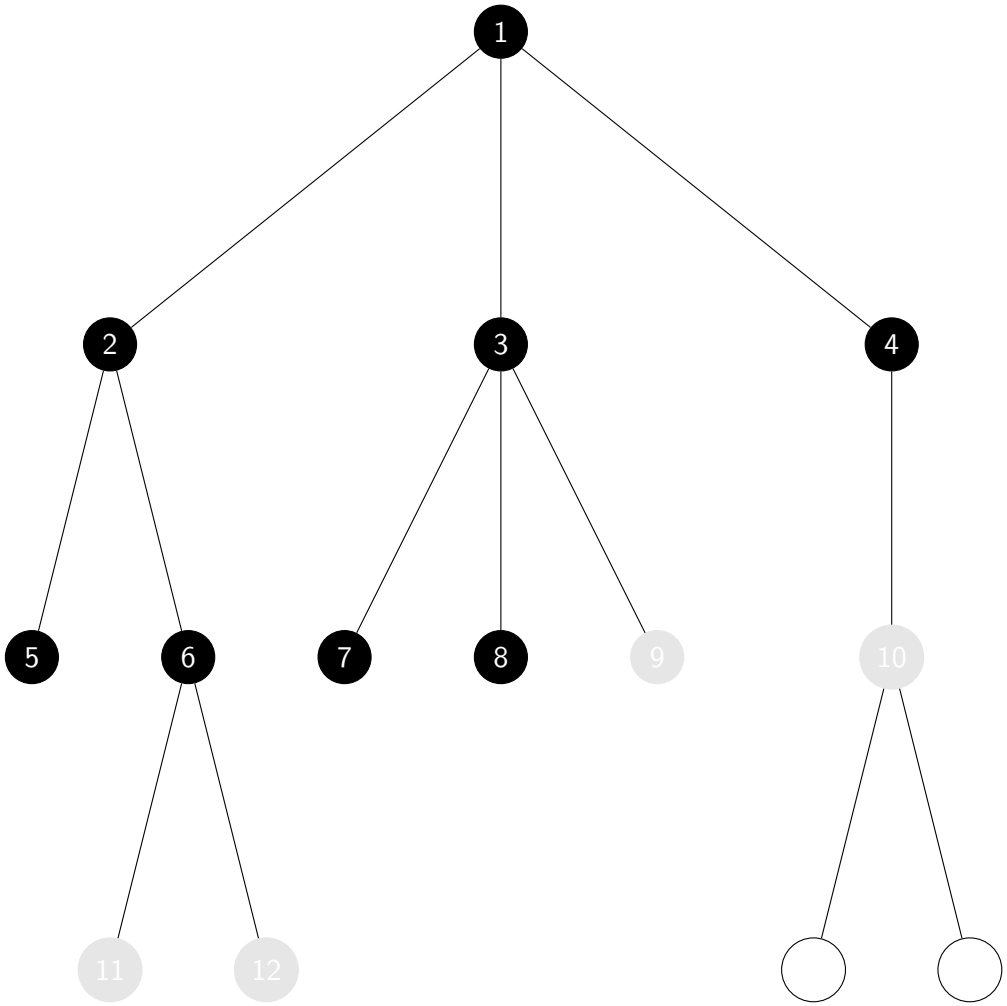
Parcours en largeur d'un arbre



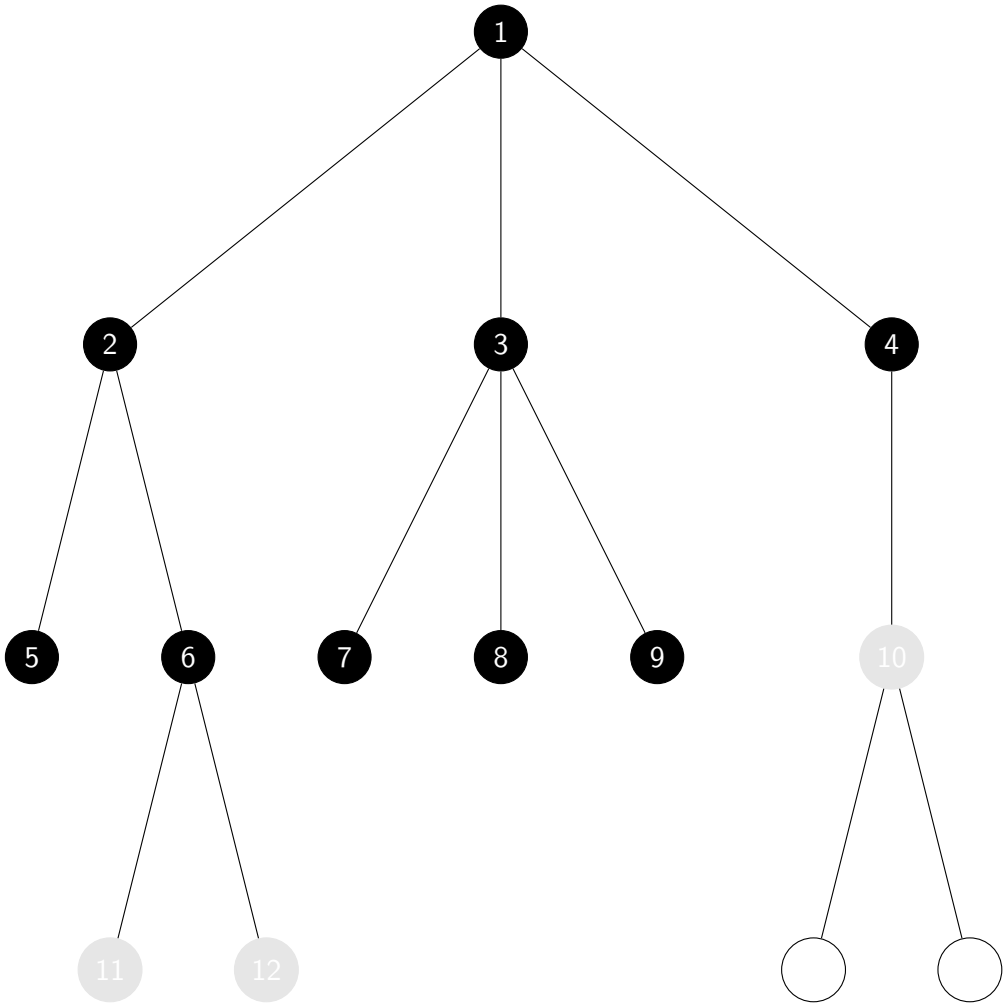
Parcours en largeur d'un arbre



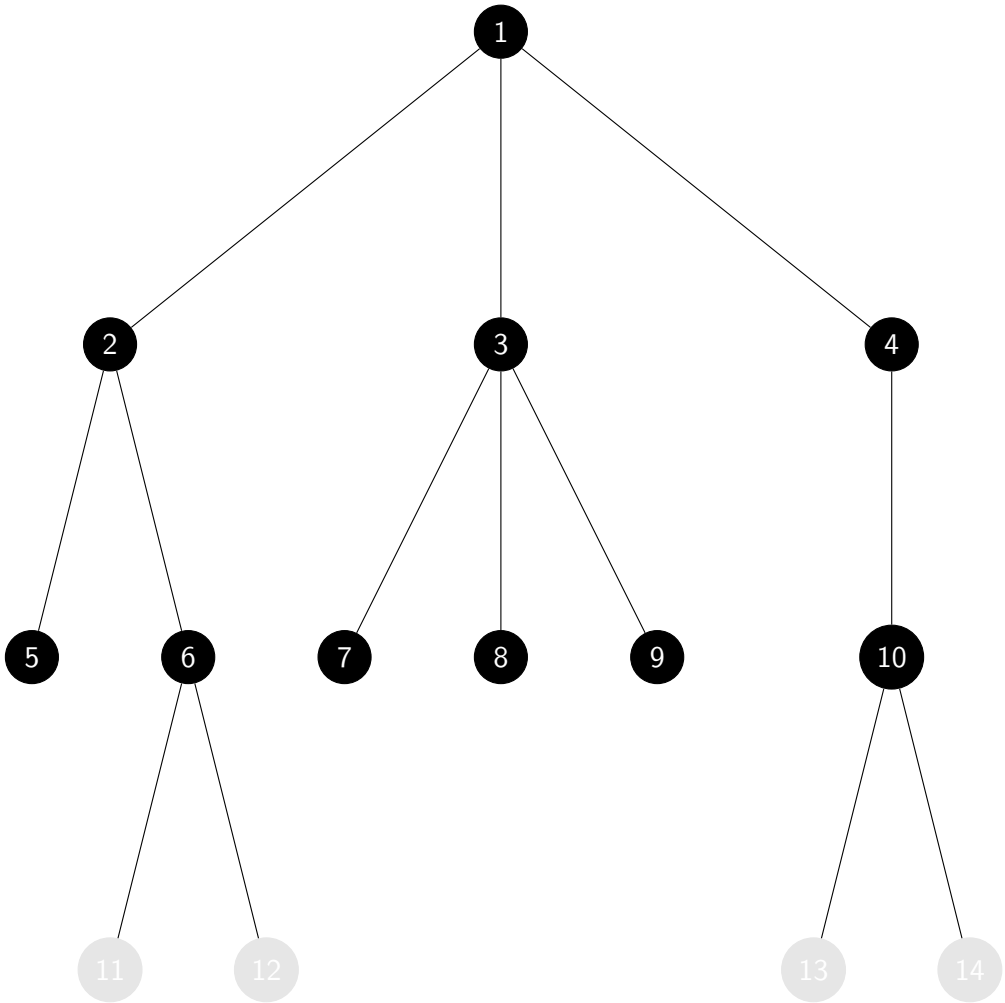
Parcours en largeur d'un arbre



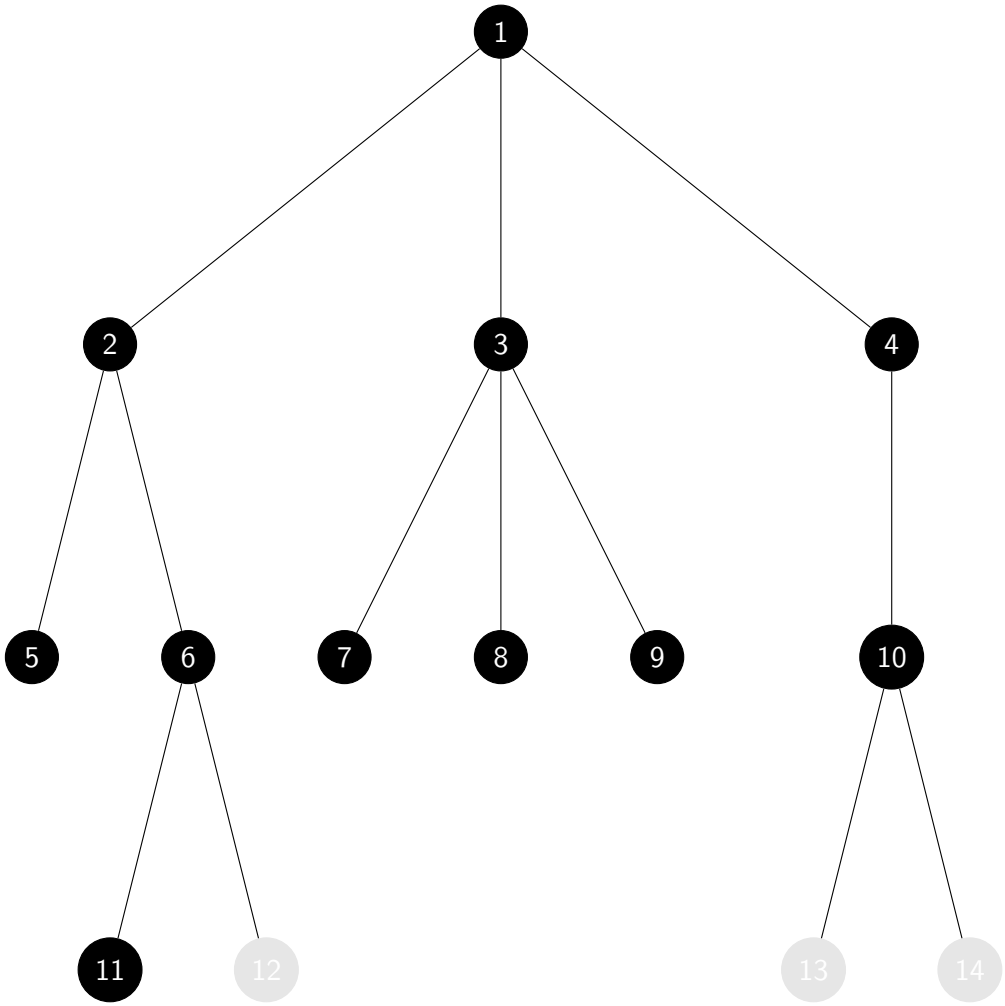
Parcours en largeur d'un arbre



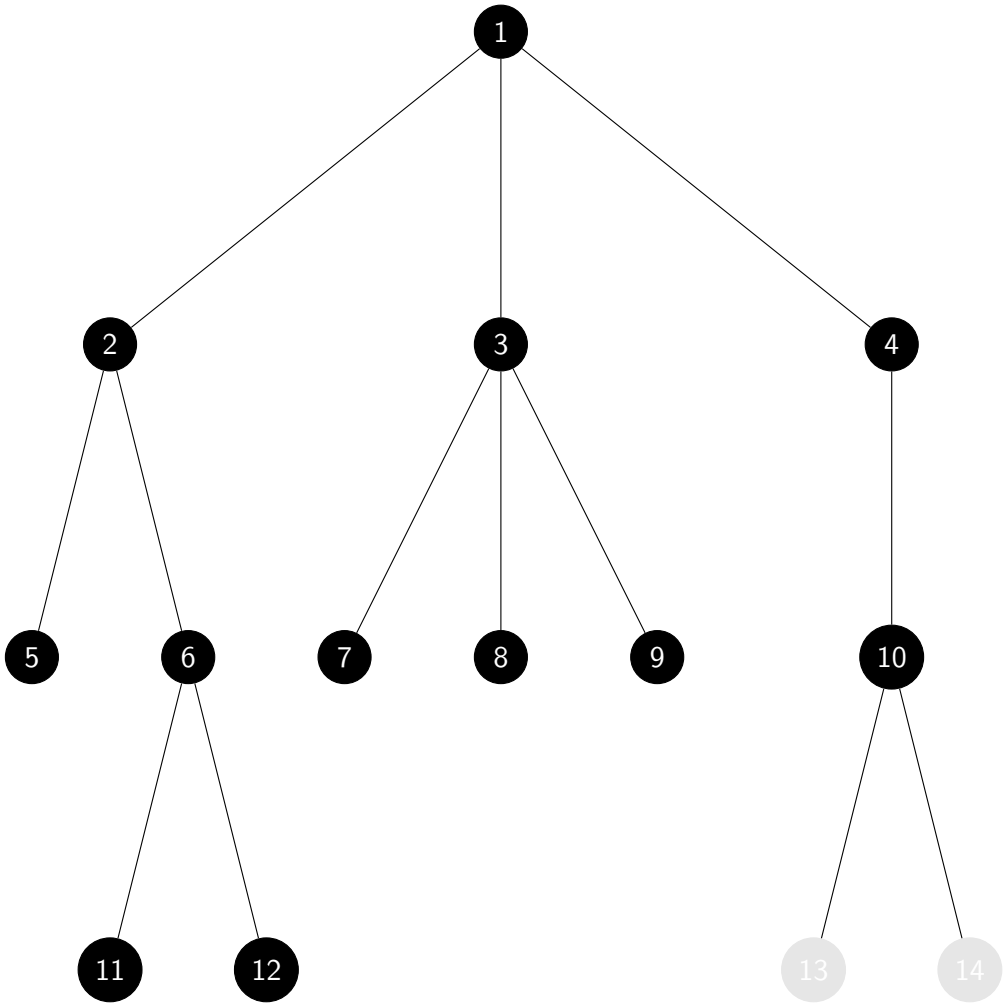
Parcours en largeur d'un arbre



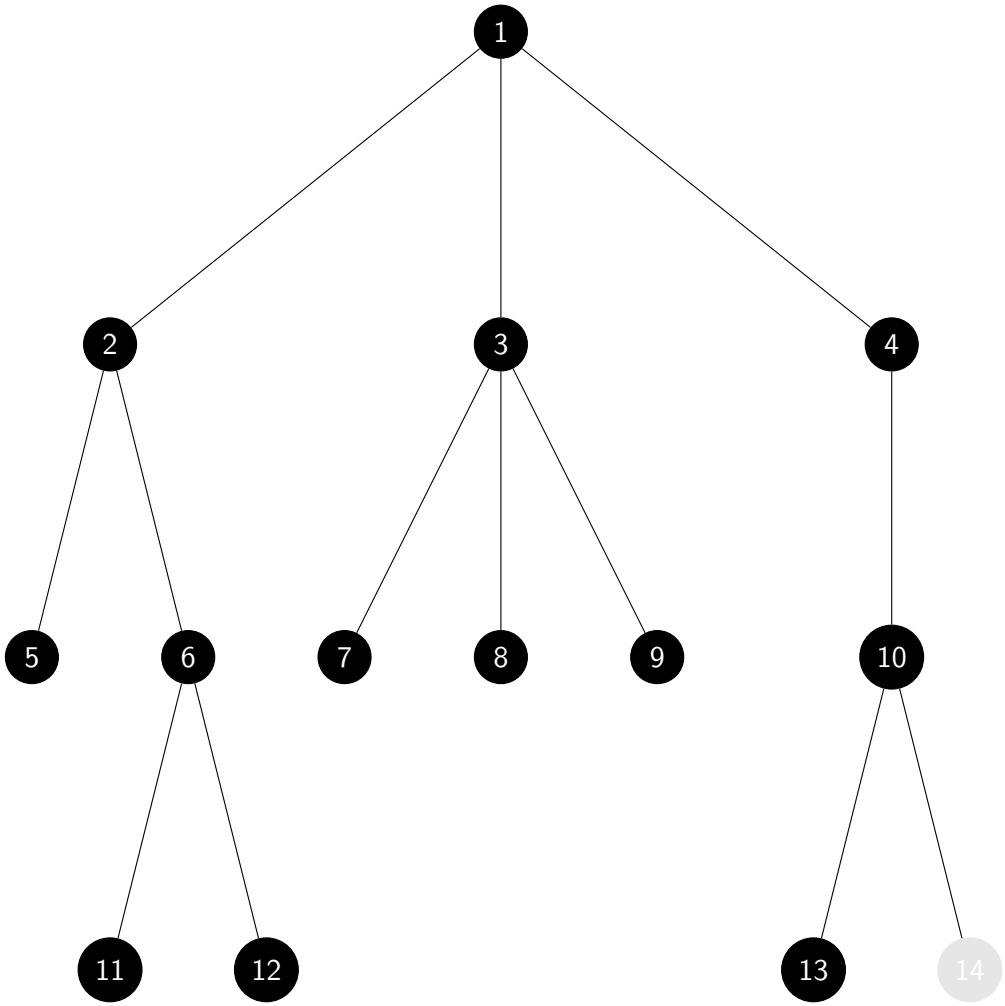
Parcours en largeur d'un arbre



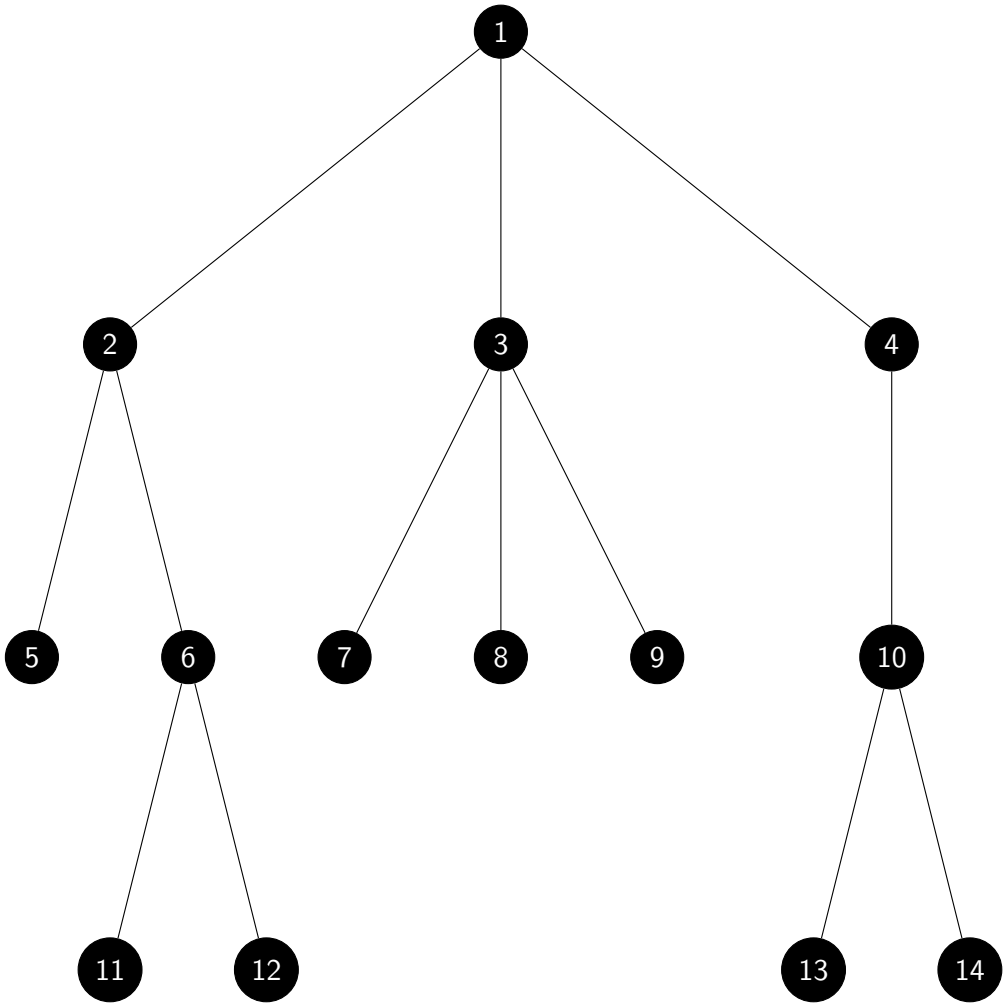
Parcours en largeur d'un arbre



Parcours en largeur d'un arbre



Parcours en largeur d'un arbre



Parcours en largeur : une propriété

Une façon de comprendre l'algorithme est d'utiliser une notion de distance :

Parcours en largeur : une propriété

Une façon de comprendre l'algorithme est d'utiliser une notion de distance :

- une page est à distance 1 de la page de départ si on l'atteint par un lien direct depuis la page 1,

Parcours en largeur : une propriété

Une façon de comprendre l'algorithme est d'utiliser une notion de distance :

- une page est à distance 1 de la page de départ si on l'atteint par un lien direct depuis la page 1,
- elle est à distance 2 de la page de départ si on peut l'atteindre (par le plus court chemin) en passant par une page à distance 1 du départ,

Parcours en largeur : une propriété

Une façon de comprendre l'algorithme est d'utiliser une notion de distance :

- une page est à distance 1 de la page de départ si on l'atteint par un lien direct depuis la page 1,
- elle est à distance 2 de la page de départ si on peut l'atteindre (par le plus court chemin) en passant par une page à distance 1 du départ,
- elle est à distance 3 du départ si on peut l'atteindre (par le plus court chemin) en passant par une page à distance 1 et une page à distance 2...

Parcours en largeur : une propriété

Une façon de comprendre l'algorithme est d'utiliser une notion de distance :

- une page est à distance 1 de la page de départ si on l'atteint par un lien direct depuis la page 1,
- elle est à distance 2 de la page de départ si on peut l'atteindre (par le plus court chemin) en passant par une page à distance 1 du départ,
- elle est à distance 3 du départ si on peut l'atteindre (par le plus court chemin) en passant par une page à distance 1 et une page à distance 2...

L'algorithme de parcours en largeur va visiter en premier lieu toutes les pages à distance 1 du départ, puis toutes les pages à distance 2 du départ, puis toutes les pages à distance 3...(c'est en fait cette propriété qui donne son nom à ce type de parcours).

BFS (breadth first search) : programmation python

Exercice avec corrigé.
Avec la représentation d'un graphe par un dictionnaire comme précédemment, programmer en langage python le BFS avec les variables suivantes :

BFS (breadth first search) : programmation python

Exercice avec corrigé.
Avec la représentation d'un graphe par un dictionnaire comme précédemment, programmer en langage python le BFS avec les variables suivantes :

- Un dictionnaire P. En fin de parcours, pour tout sommet s du graphe P[s] sera le père de s, c'est à dire le sommet à partir duquel le sommet s a été découvert lors du parcours.

BFS (breadth first search) : programmation python

Exercice avec corrigé.
Avec la représentation d'un graphe par un dictionnaire comme précédemment, programmer en langage python le BFS avec les variables suivantes :

- Un dictionnaire P. En fin de parcours, pour tout sommet s du graphe P[s] sera le père de s, c'est à dire le sommet à partir duquel le sommet s a été découvert lors du parcours.
- Un dictionnaire couleur. Pour tout sommet s, couleur[s] vaut blanc si le sommet s n'est pas passé dans la file, gris s'il est dans la file, noir lorsqu'il est sorti de la file.

BFS (breadth first search) : programmation python

Exercice avec corrigé.
Avec la représentation d'un graphe par un dictionnaire comme précédemment, programmer en langage python le BFS avec les variables suivantes :

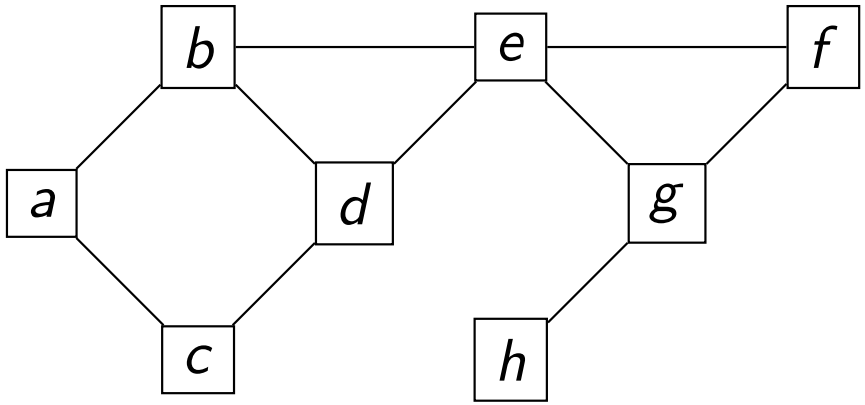
- Un dictionnaire P. En fin de parcours, pour tout sommet s du graphe P[s] sera le père de s, c'est à dire le sommet à partir duquel le sommet s a été découvert lors du parcours.
- Un dictionnaire couleur. Pour tout sommet s, couleur[s] vaut blanc si le sommet s n'est pas passé dans la file, gris s'il est dans la file, noir lorsqu'il est sorti de la file.
- Une liste Q utilisée comme file (fifo) : on enfile un sommet lorsqu'il est découvert, on le défile lorsqu'il est terminé (traitement prioritaire des sommets découverts au plus tôt).

Exemple d'exécution

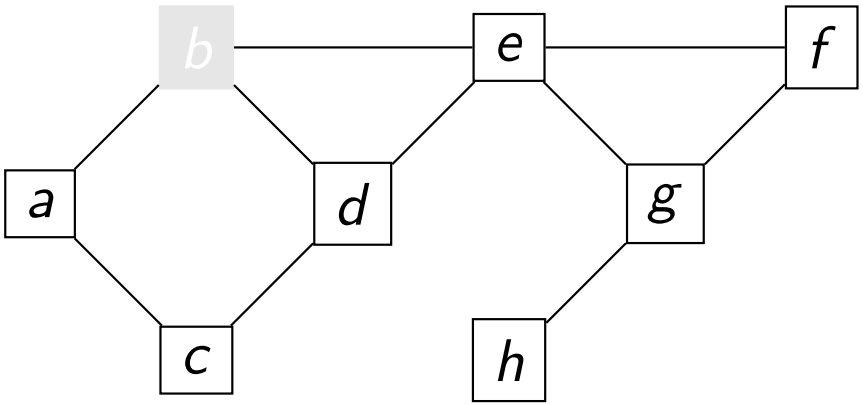
Déroulement attendu du programme, avec l'appel bfs(G,'b'), sur le graphe ci-dessous :

Python

```
G=dict()
G['a']=['b','c']
G['b']=['a','d','e']
G['c']=['a','d']
G['d']=['b','c','e']
G['e']=['b','d','f','g']
G['f']=['e','g']
G['g']=['e','f','h']
G['h']=['g']
```

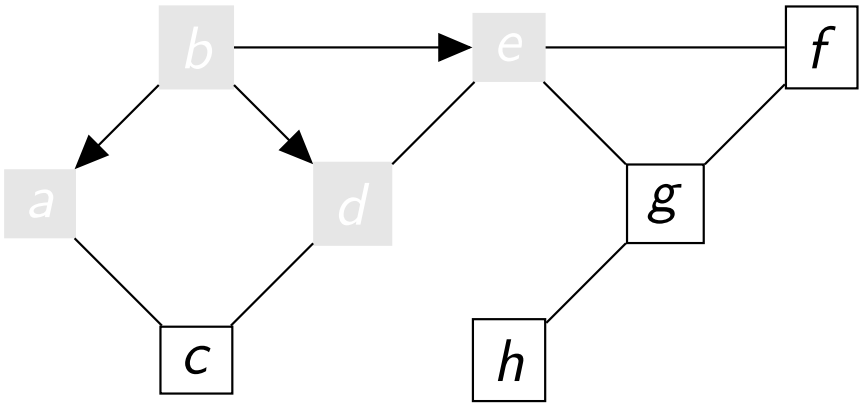


Déroulement



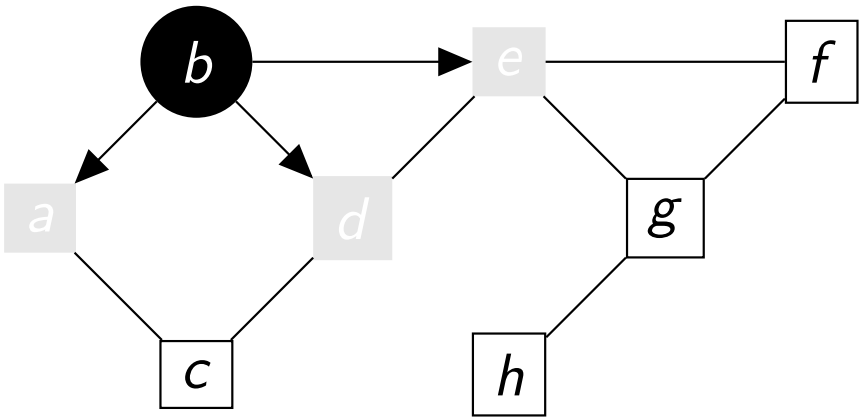
P={ 'b' : None}
Q=['b']
Découverts (gris ou noirs) =['b']
Fermés (noirs) =[]

Déroulement



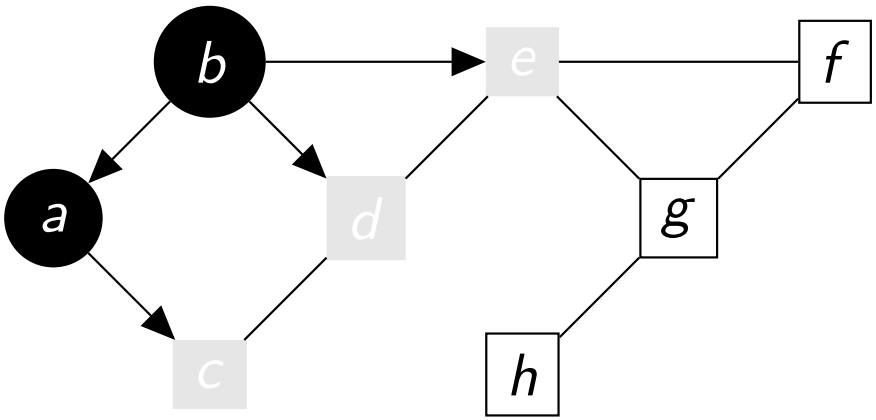
P={ 'b' : None, 'a' :'b', 'd' :'b', 'e' :'b'}
Q=['b','a','d','e']
Découverts =['b','a','d','e']
Fermés=[]

Déroulement



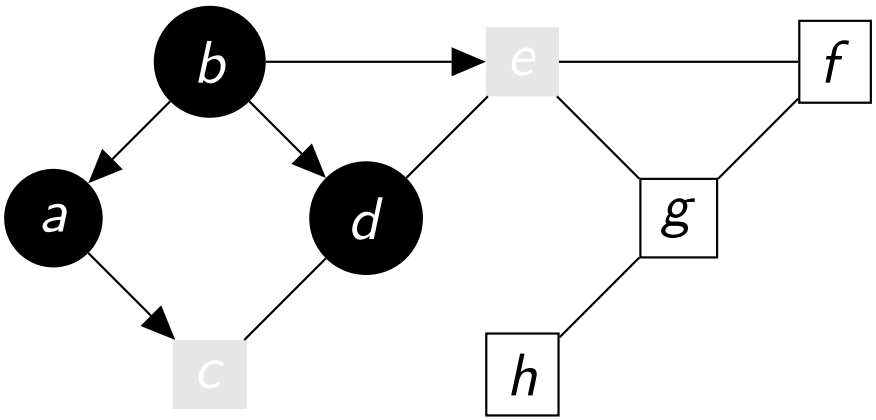
P={ 'b' : None, 'a' :'b', 'd' :'b', 'e' :'b'}
Q=['a','d','e']
Découverts=['b','a','d','e']
Fermés=['b']

Déroulement



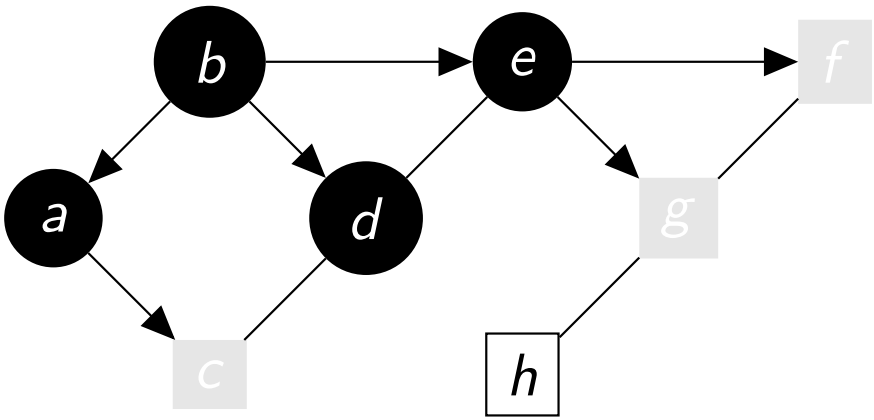
P={ 'b' : None, 'a' :'b', 'd' :'b', 'e' :'b','c' :'a'}
Q=['d','e','c']
Découverts=['b','a','d','e','c']
Fermés=['b','a']

Déroulement



P={ 'b' : None, 'a' :'b', 'd' :'b', 'e' :'b','c' :'a'}
Q=['e','c']
Découverts=['b','a','d','e','c']
Fermés=['b','a','d']

Déroulement



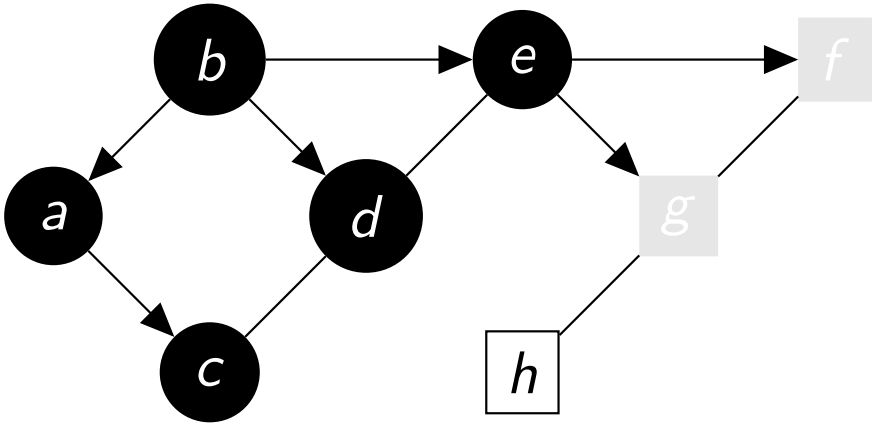
P={ 'b' : None, 'a' :'b', 'd' :'b', 'e' :'b','c' :'a','f' :'e','g' :'e'}

Q=['c','f','g']

Découverts=['b','a','d','e','c','f','g']

Fermés=['b','a','d','e']

Déroulement



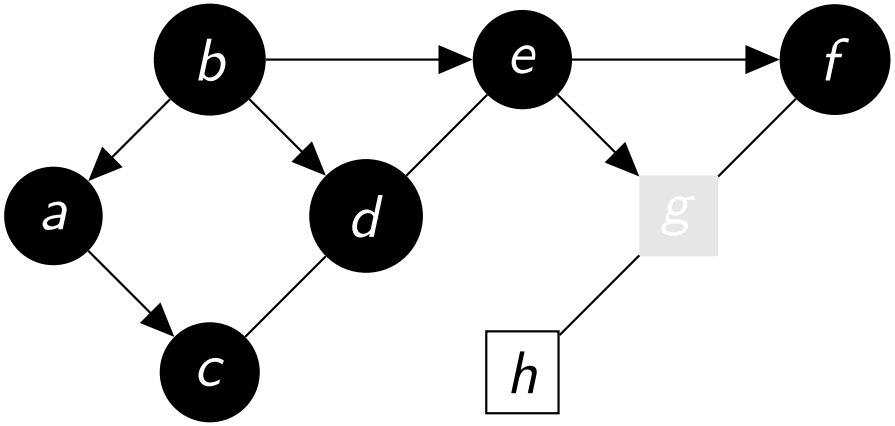
P={ 'b' : None, 'a' :'b', 'd' :'b', 'e' :'b','c' :'a','f' :'e','g' :'e'}

Q=['f','g']

Découverts=['b','a','d','e','c','f','g']

Fermés=['b','a','d','e','c']

Déroulement



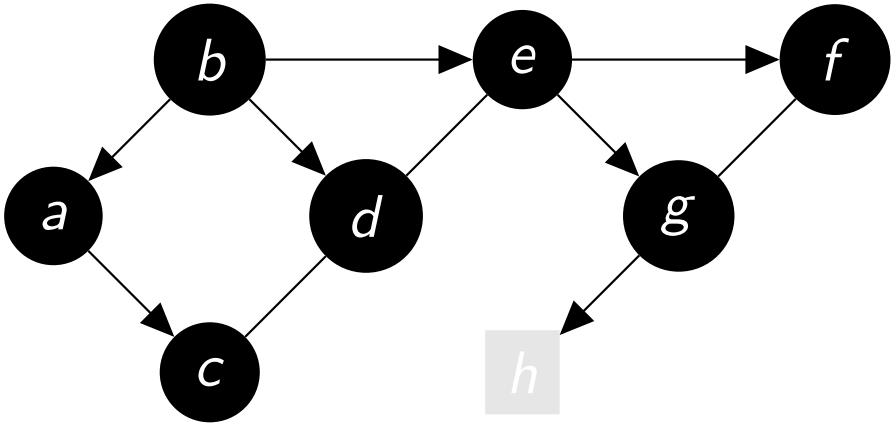
P={ 'b' : None, 'a' :'b', 'd' :'b', 'e' :'b','c' :'a','f' :'e','g' :'e'}

Q=['g']

Découverts=['b','a','d','e','c','f','g']

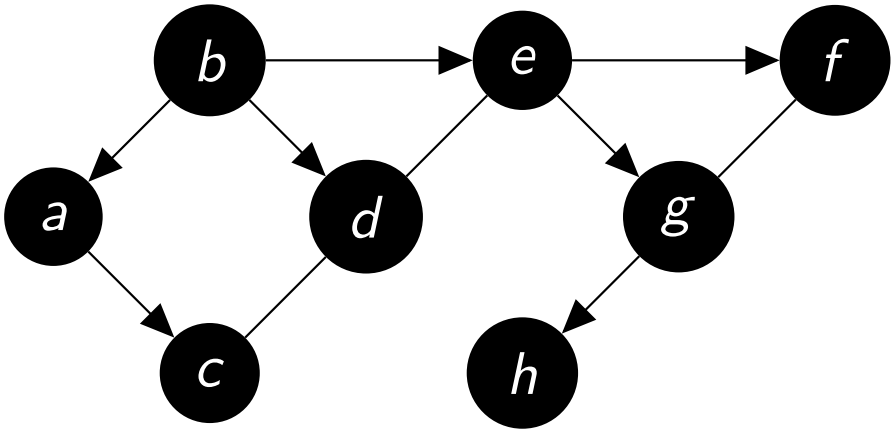
Fermés=['b','a','d','e','c','f']

Déroulement



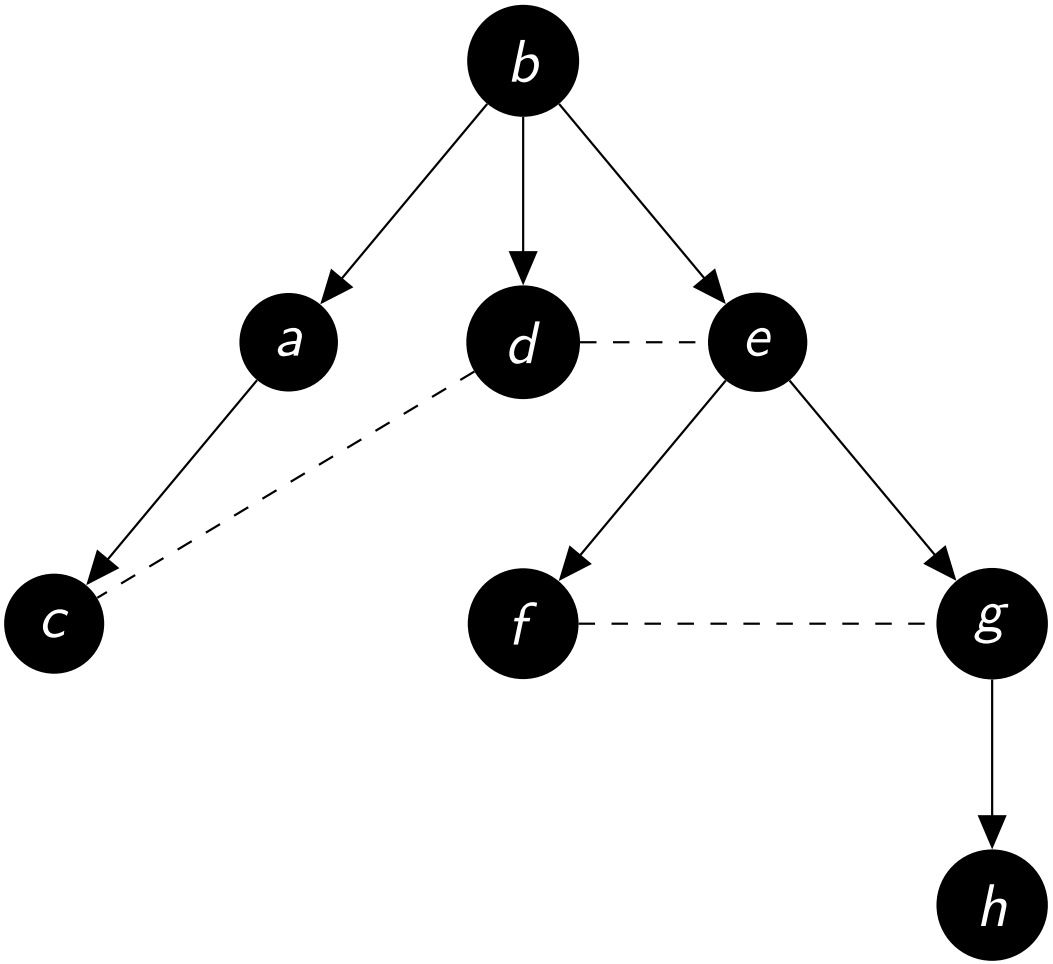
P={ 'b' : None, 'a' :'b', 'd' :'b', 'e' :'b','c' :'a','f' :'e','g' :'e','h' :'g'}
Q=['h']
Découverts=['b','a','d','e','c','f','g','h']
Fermés=['b','a','d','e','c','f','g']

Déroulement



P={ 'b' : None, 'a' :'b', 'd' :'b', 'e' :'b','c' :'a','f' :'e','g' :'e','h' :'g'}
Q=[]
Découverts=['b','a','d','e','c','f','g','h']
Fermés=['b','a','d','e','c','f','g','h']

Arborescence associée au parcours



L'ordre de parcours est : ligne après ligne (de la racine vers les feuilles) et de gauche à droite pour une ligne.

BFS : un codage en python

Python

```
def bfs(G,s) :
    couleur=dict()
    for x in G : couleur[x]='blanc'
    P=dict()
    P[s]=None
    couleur[s]='gris'
    Q=[s]
    while Q :
        u=Q[0]
        for v in G[u] :
            if couleur[v]=='blanc' :
                P[v]=u
                couleur[v]='gris'
                Q.append(v)
        Q.pop(0)
        couleur[u]='noir'
    return P
```

BFS en python

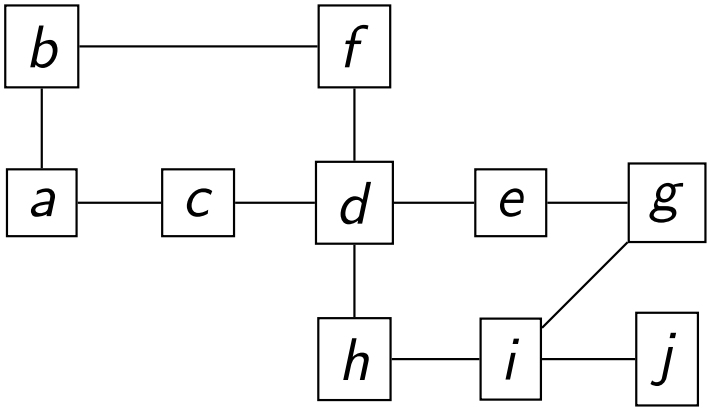
En ne gardant que l'essentiel

Python

```
def bfs(G,s) :
    P,Q={s :None},[s]
    while Q :
        u=Q.pop(0)
        for v in G[u] :
            if v in P : continue
            P[v]=u
            Q.append(v)
    return P
```

Exercice à rendre 1

Vous donnerez, sur papier, un codage python du graphe ci-dessous suivant le modèle précédemment donné puis le déroulement de bfs(G,'b') correspondant à ce codage :



Application

Exercice à rendre 2. En utilisant le programme précédent, on définit la fonction suivante :

Python

```
def f(G,s,v) :
    P=bfs(G,s)
    ch=[v]
    while P[v] :
        ch.append(P[v])
        v=P[v]
    ch.reverse()
    return ch
print f(G,'b','h')
```

Quel est son rôle ?

Parcours en profondeur

Parcours en profondeur : principe de l'algorithme

Vous devez parcourir toutes les pages d'un site web. Les pages sont les sommets d'un graphe et un lien entre deux pages est une arête entre ces deux sommets.

Parcours en profondeur : principe de l'algorithme

Vous devez parcourir toutes les pages d'un site web. Les pages sont les sommets d'un graphe et un lien entre deux pages est une arête entre ces deux sommets.

1. Dans le parcours en profondeur, on utilise une pile. On empile le sommet de départ (on visite la page index du site).

Parcours en profondeur : principe de l'algorithme

Vous devez parcourir toutes les pages d'un site web. Les pages sont les sommets d'un graphe et un lien entre deux pages est une arête entre ces deux sommets.

1. Dans le parcours en profondeur, on utilise une pile. On empile le sommet de départ (on visite la page index du site).
2. Si le sommet de la pile présente des voisins qui ne sont pas dans la pile, ni déjà passés dans la pile :

Parcours en profondeur : principe de l'algorithme

Vous devez parcourir toutes les pages d'un site web. Les pages sont les sommets d'un graphe et un lien entre deux pages est une arête entre ces deux sommets.

1. Dans le parcours en profondeur, on utilise une pile. On empile le sommet de départ (on visite la page index du site).
2. Si le sommet de la pile présente des voisins qui ne sont pas dans la pile, ni déjà passés dans la pile :
   - alors on sélectionne l'un de ces voisins et on l'empile (en le marquant de son numéro de découverte),

Parcours en profondeur : principe de l'algorithme

Vous devez parcourir toutes les pages d'un site web. Les pages sont les sommets d'un graphe et un lien entre deux pages est une arête entre ces deux sommets.

1. Dans le parcours en profondeur, on utilise une pile. On empile le sommet de départ (on visite la page index du site).
2. Si le sommet de la pile présente des voisins qui ne sont pas dans la pile, ni déjà passés dans la pile :
   - alors on sélectionne l'un de ces voisins et on l'empile (en le marquant de son numéro de découverte),
   - sinon on dépile (c'est à dire on supprime l'élément du sommet de la pile).

Parcours en profondeur : principe de l'algorithme

Vous devez parcourir toutes les pages d'un site web. Les pages sont les sommets d'un graphe et un lien entre deux pages est une arête entre ces deux sommets.

1. Dans le parcours en profondeur, on utilise une pile. On empile le sommet de départ (on visite la page index du site).
2. Si le sommet de la pile présente des voisins qui ne sont pas dans la pile, ni déjà passés dans la pile :
   - alors on sélectionne l'un de ces voisins et on l'empile (en le marquant de son numéro de découverte),
   - sinon on dépile (c'est à dire on supprime l'élément du sommet de la pile).
3. On recommence au point 2 (tant que la pile n'est pas vide).

Parcours en profondeur : principe de l'algorithme

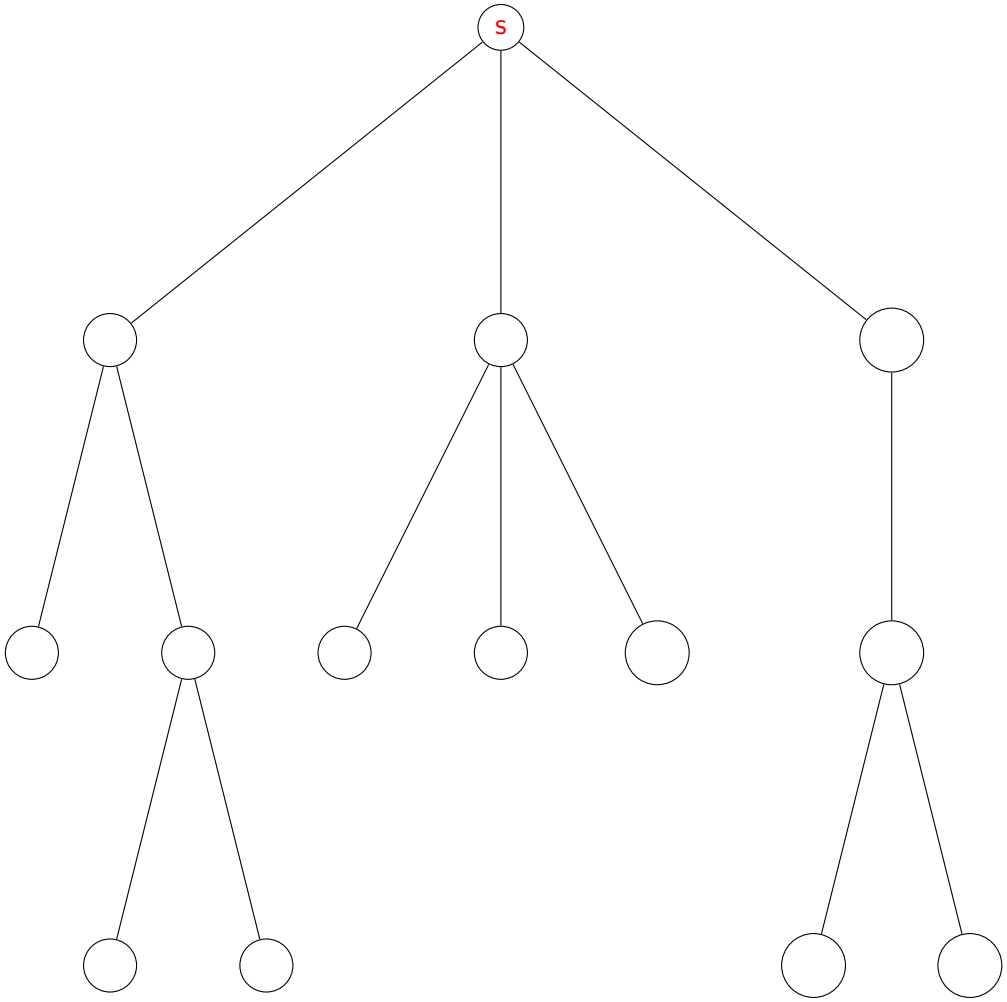
Vous devez parcourir toutes les pages d'un site web. Les pages sont les sommets d'un graphe et un lien entre deux pages est une arête entre ces deux sommets.

1. Dans le parcours en profondeur, on utilise une pile. On empile le sommet de départ (on visite la page index du site).
2. Si le sommet de la pile présente des voisins qui ne sont pas dans la pile, ni déjà passés dans la pile :
   - alors on sélectionne l'un de ces voisins et on l'empile (en le marquant de son numéro de découverte),
   - sinon on dépile (c'est à dire on supprime l'élément du sommet de la pile).
3. On recommence au point 2 (tant que la pile n'est pas vide).

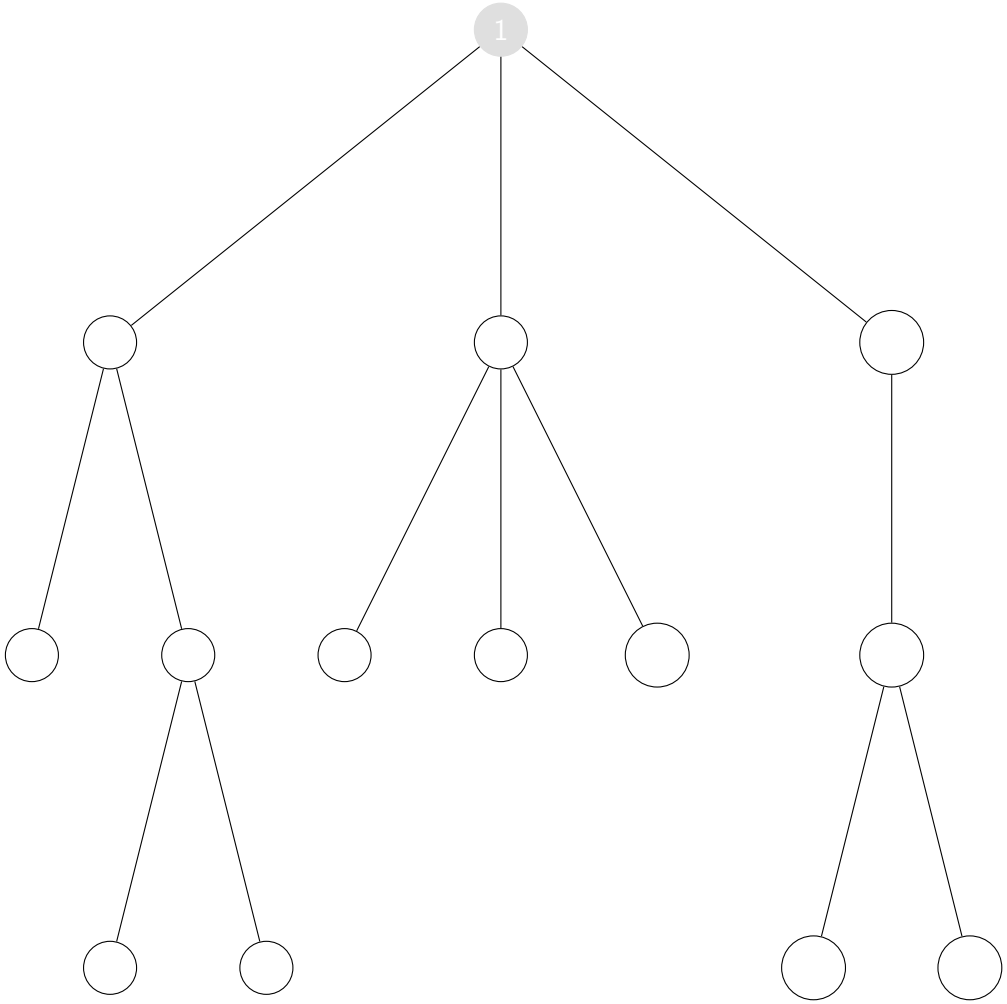
En d'autres termes, on traite toujours en priorité les liens des pages les plus tard découvertes.

Parcours en profondeur d'un arbre

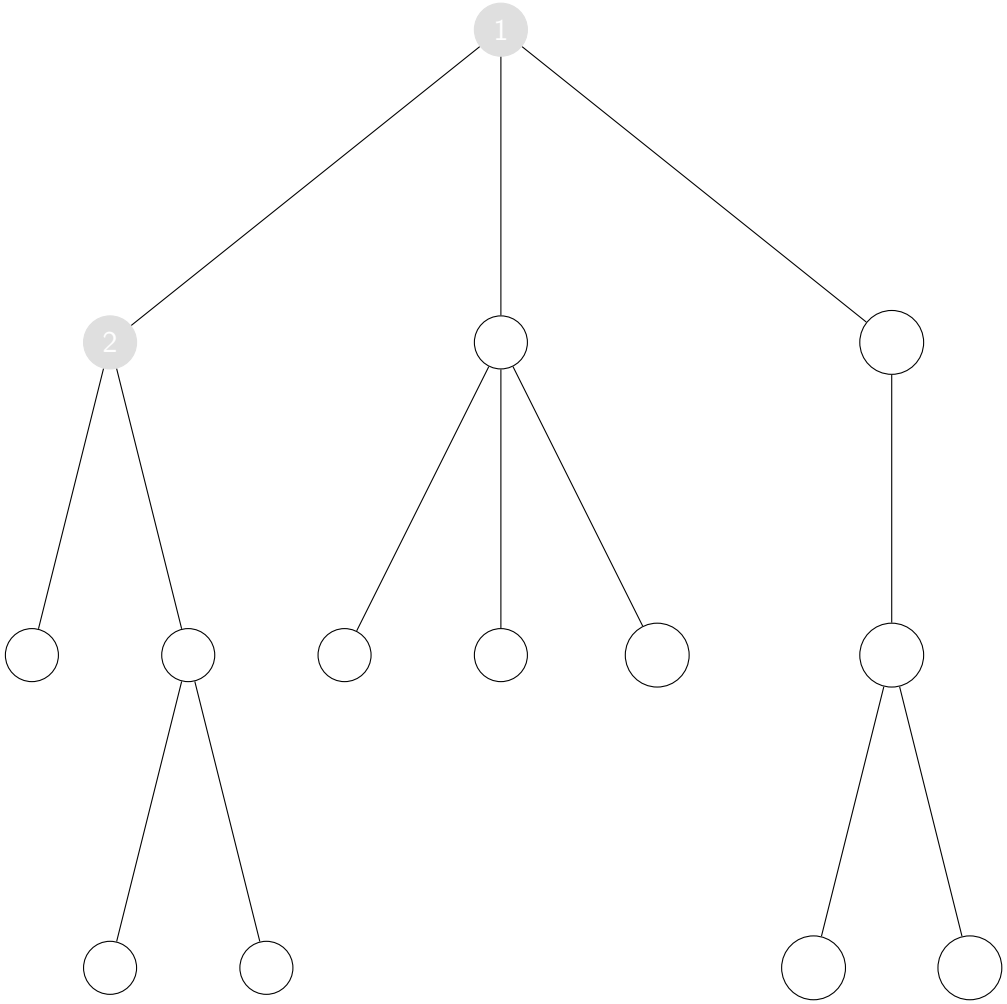
Parcourir en profondeur le graphe ci-dessous à partir du sommet s :



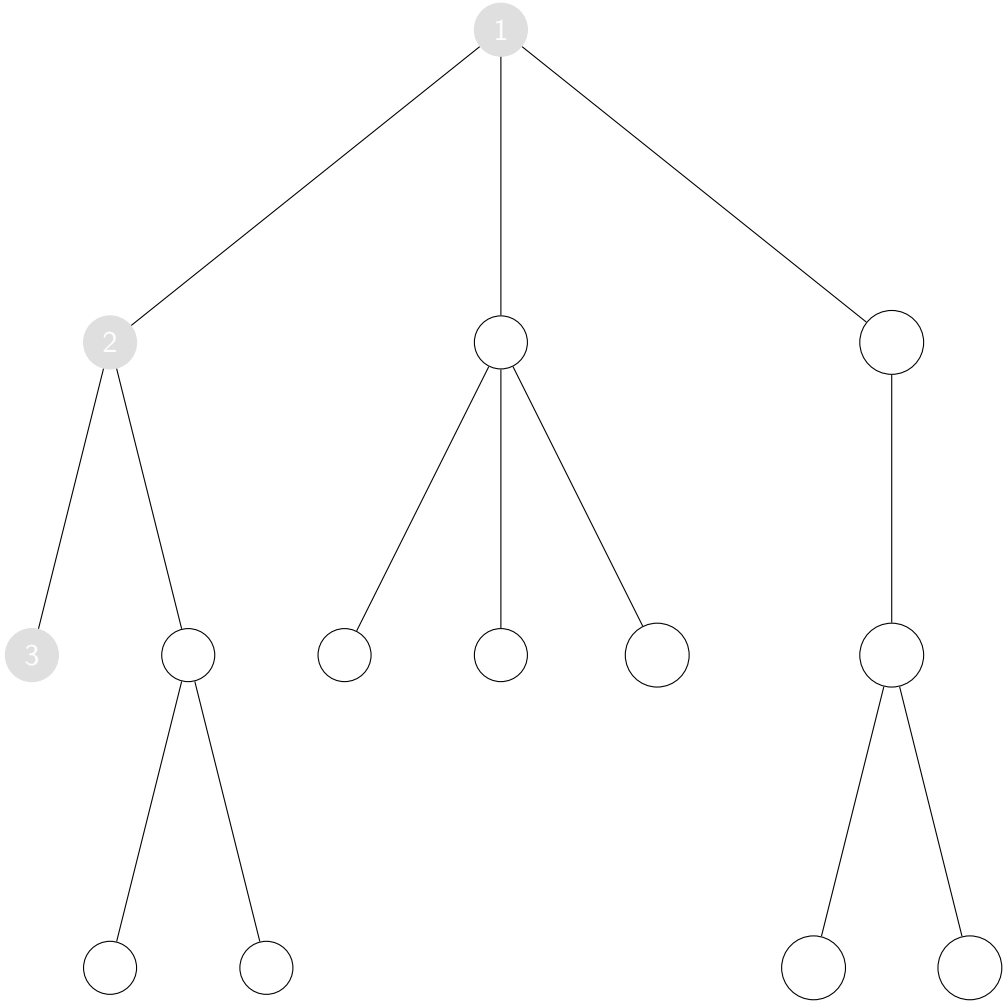
Parcours en profondeur d'un arbre



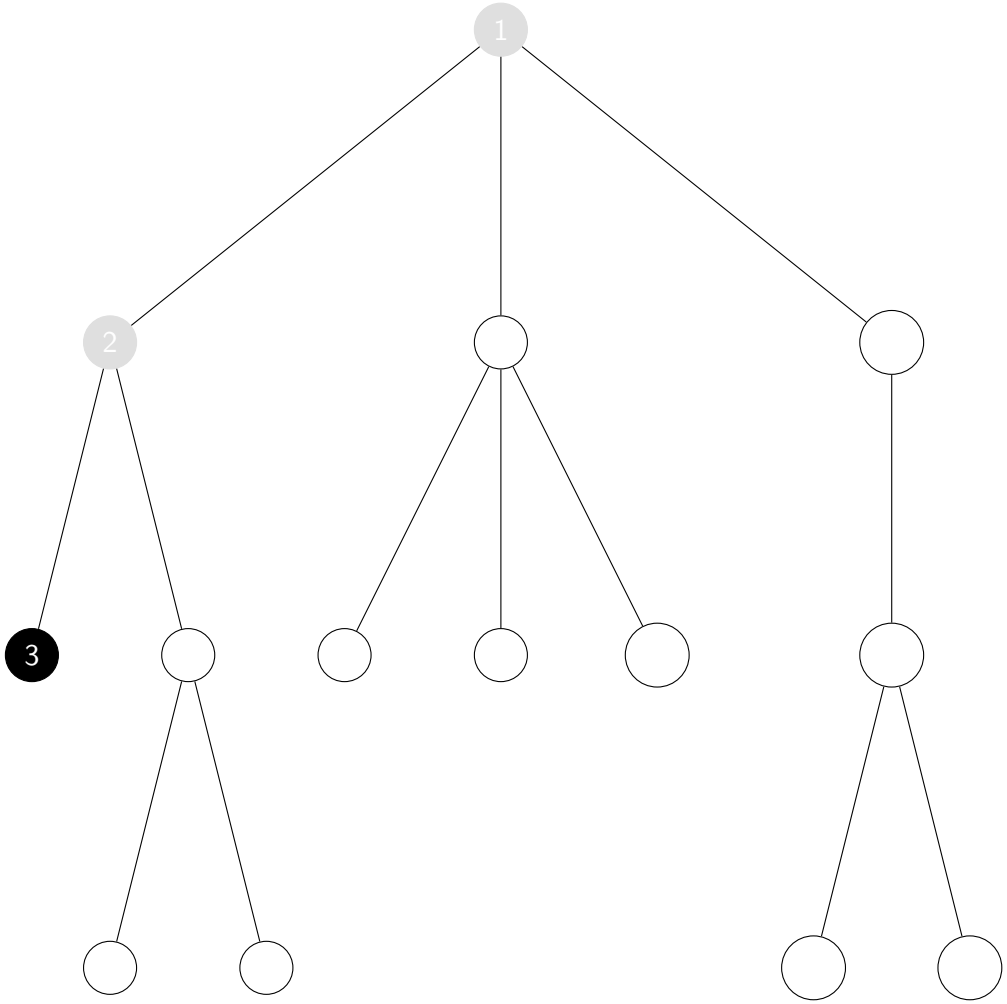
Parcours en profondeur d'un arbre



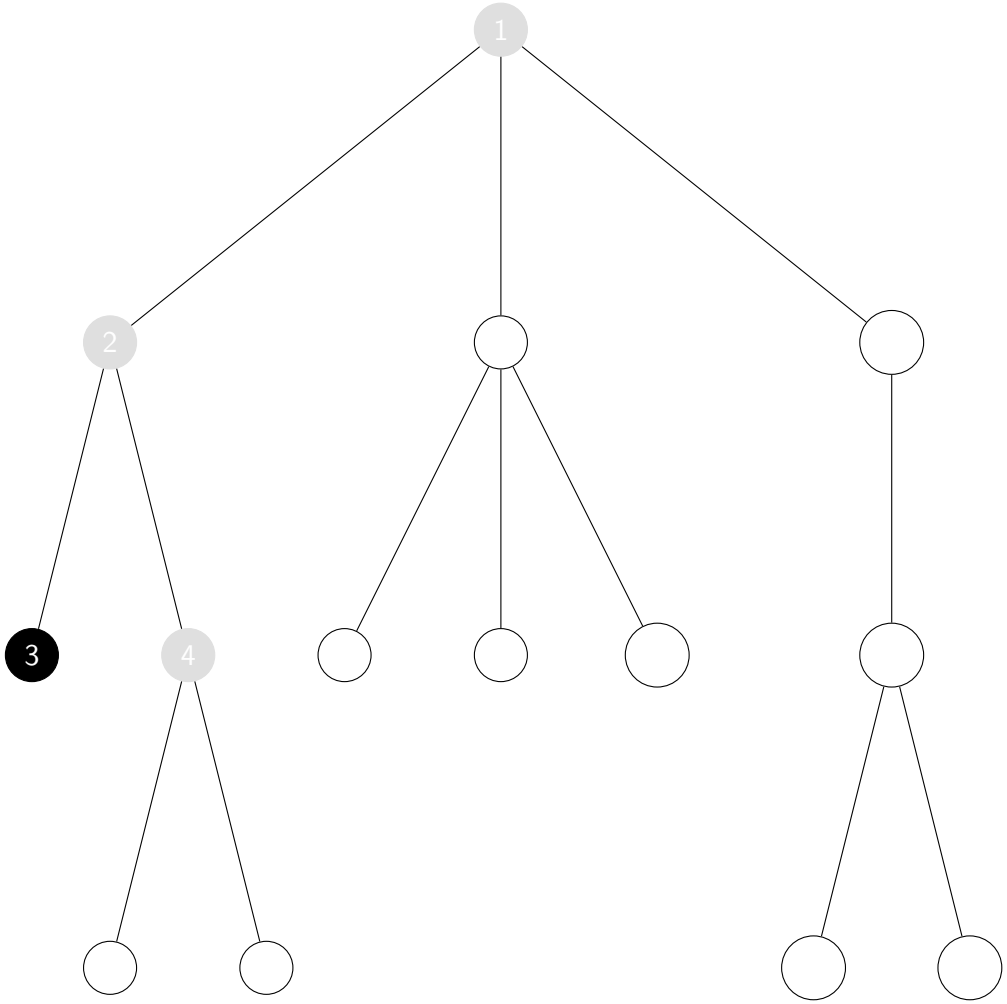
Parcours en profondeur d'un arbre



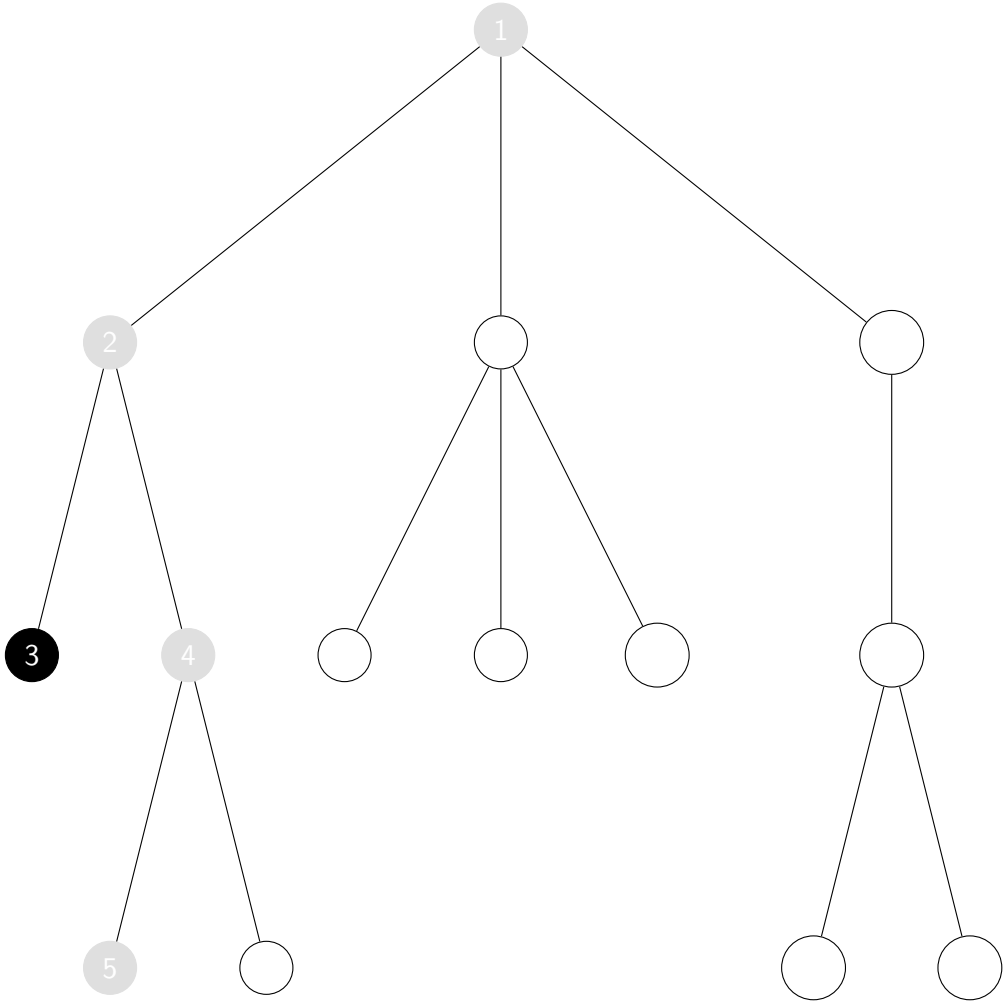
Parcours en profondeur d'un arbre



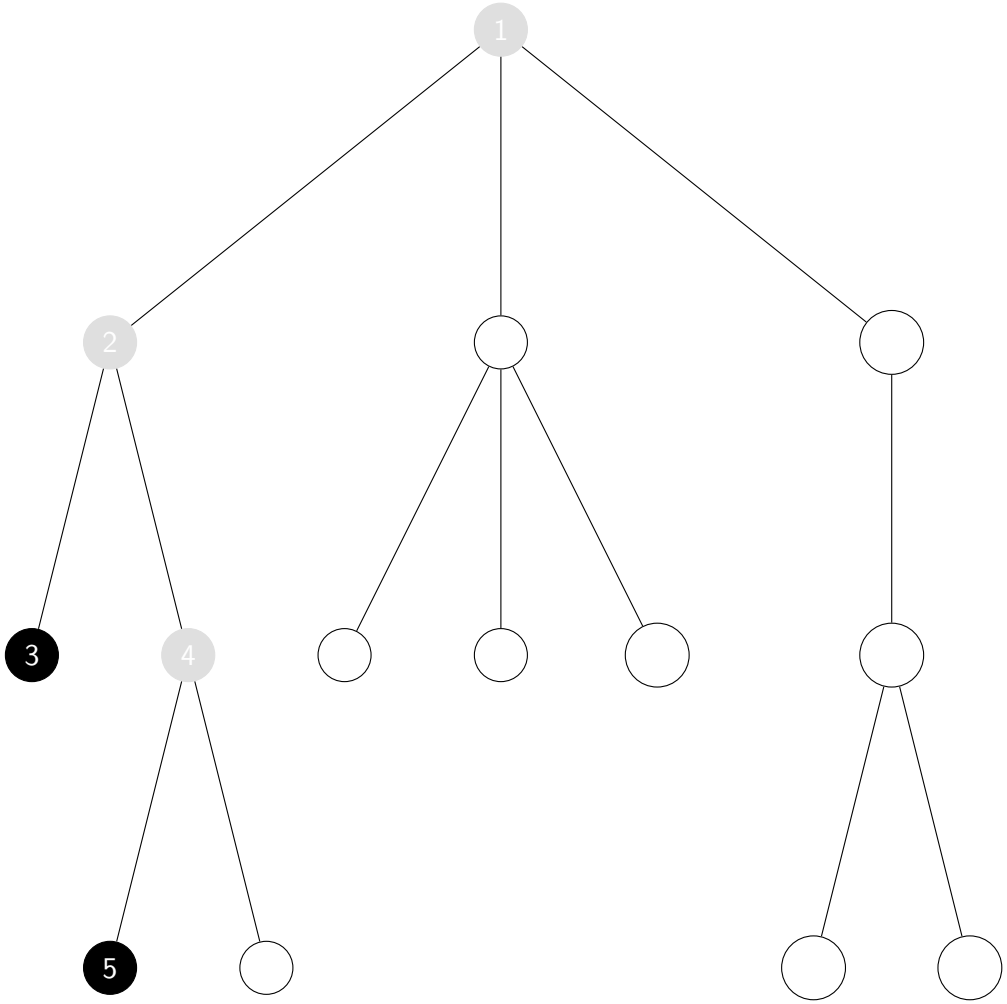
Parcours en profondeur d'un arbre



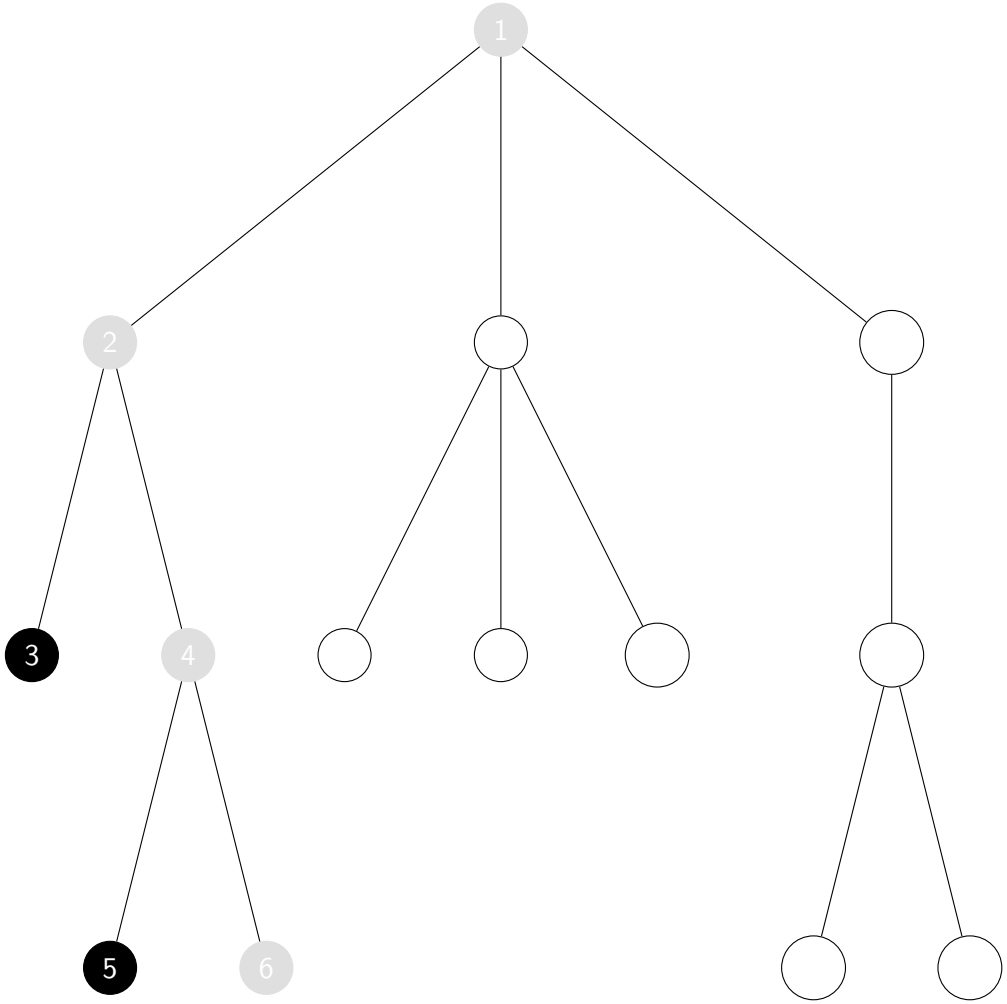
Parcours en profondeur d'un arbre



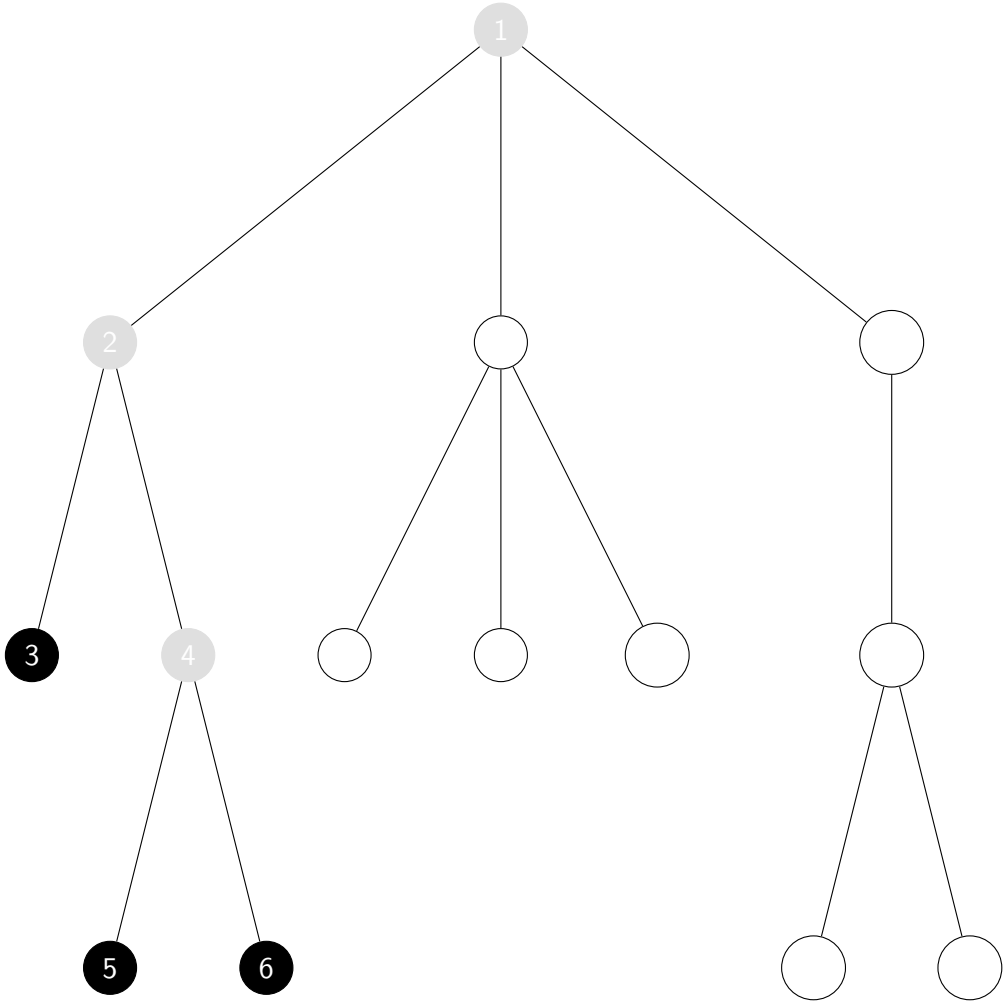
Parcours en profondeur d'un arbre



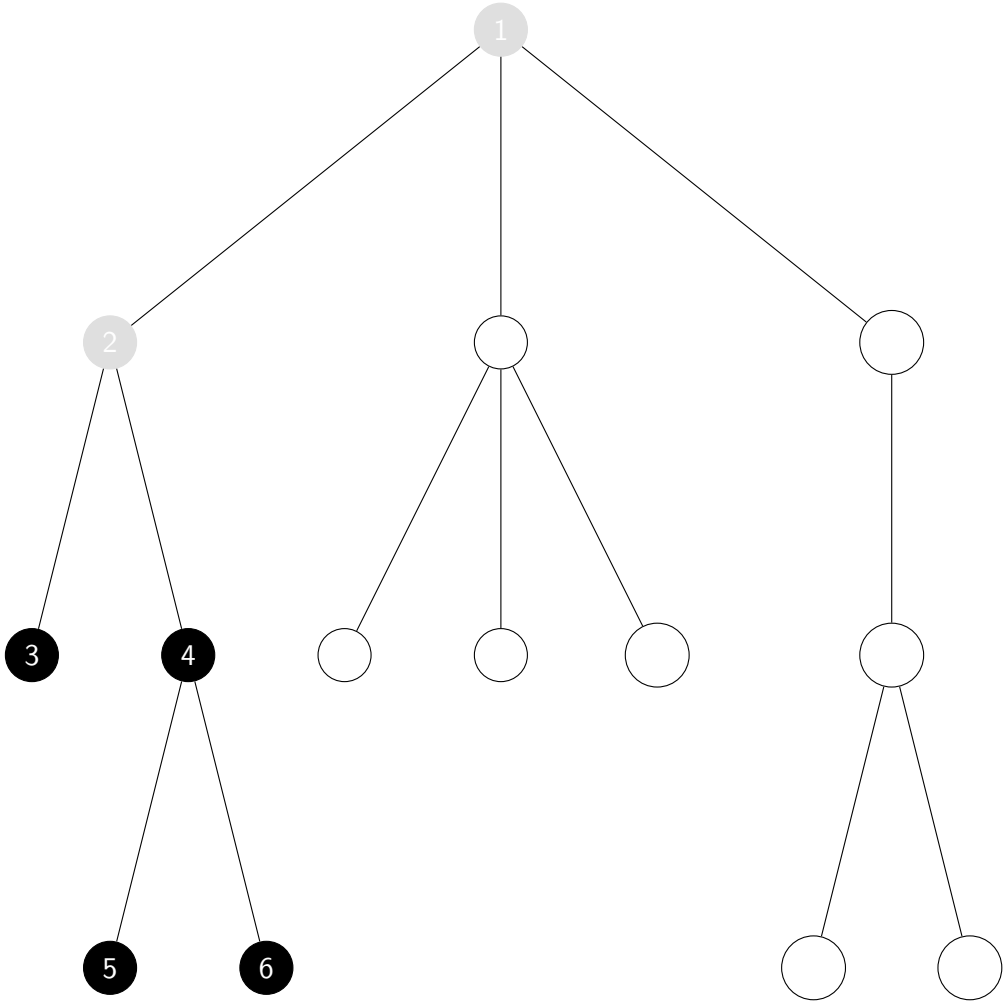
Parcours en profondeur d'un arbre



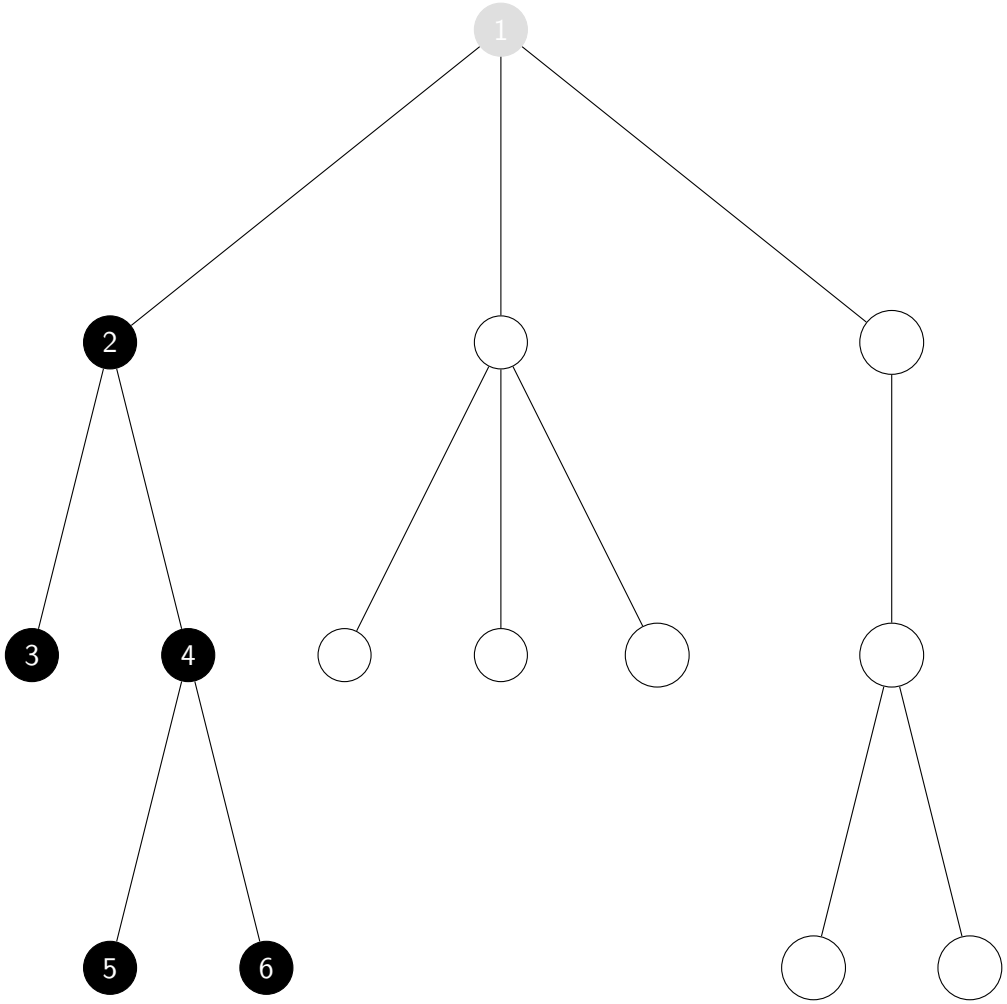
Parcours en profondeur d'un arbre



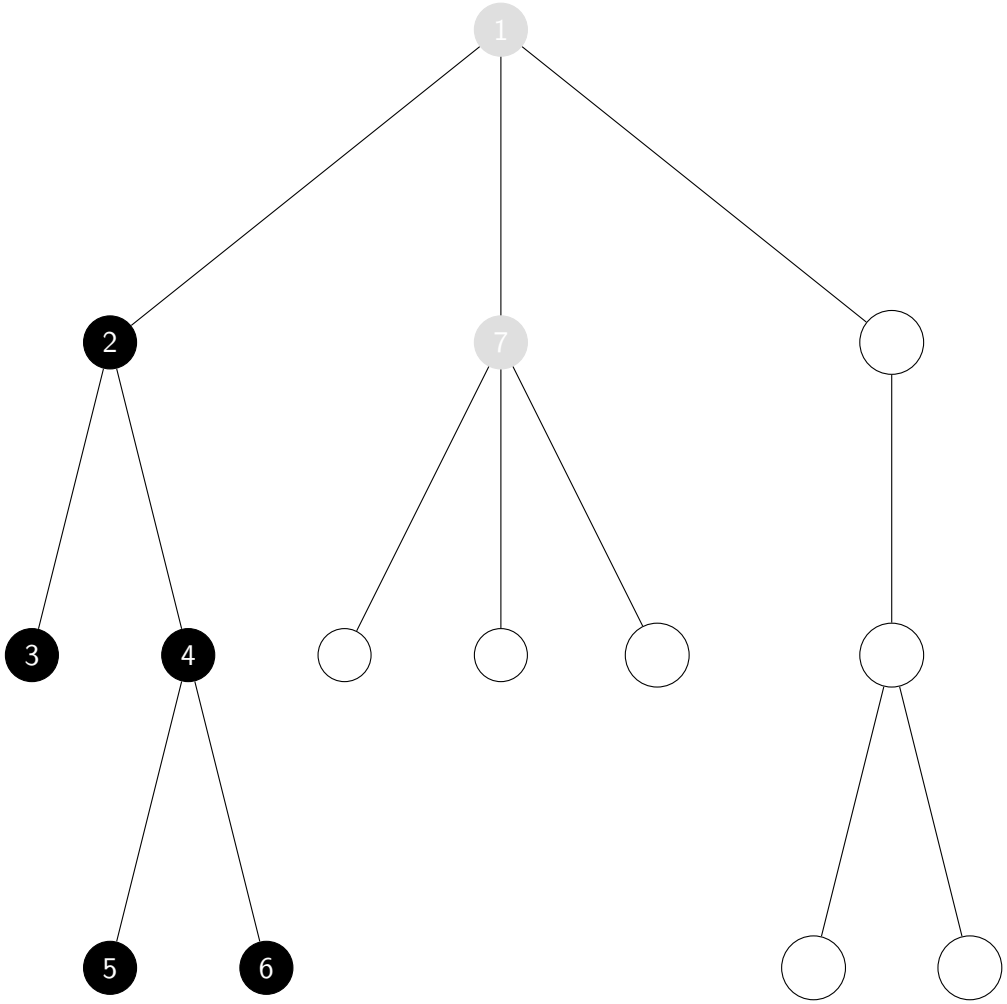
Parcours en profondeur d'un arbre



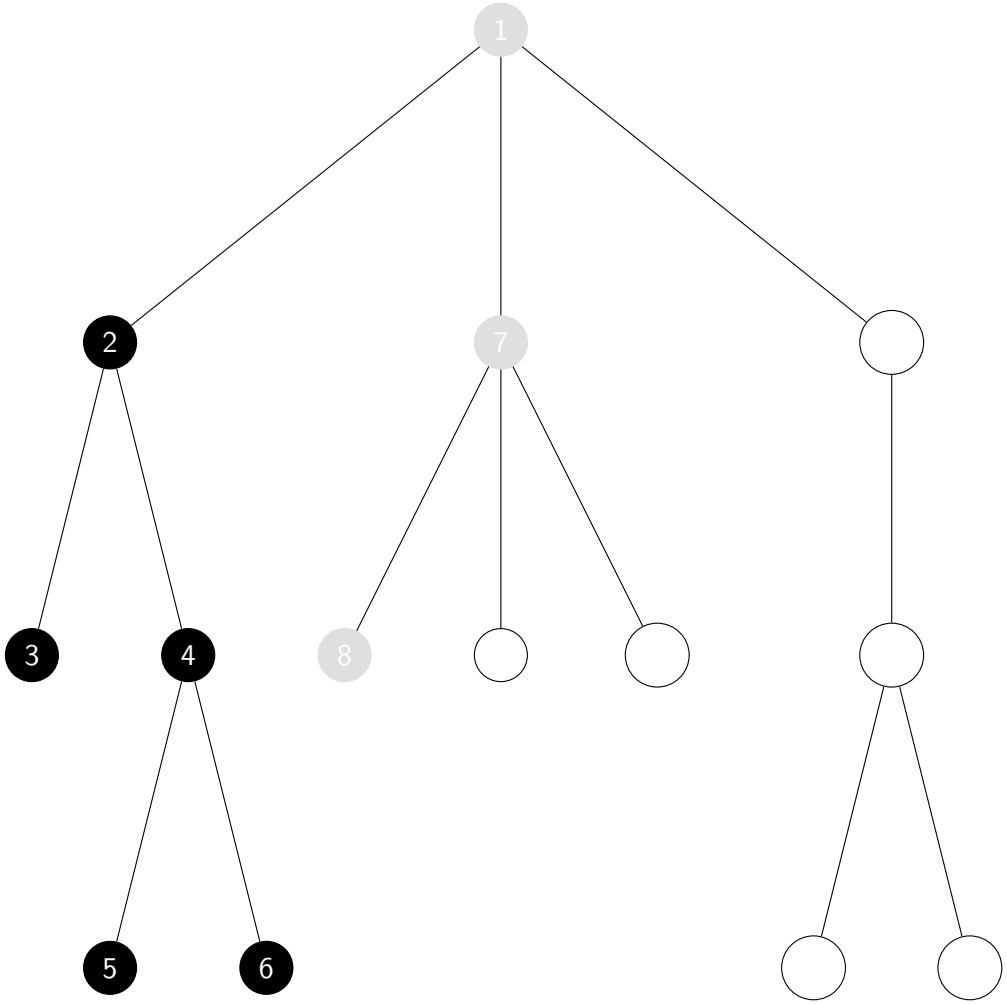
Parcours en profondeur d'un arbre



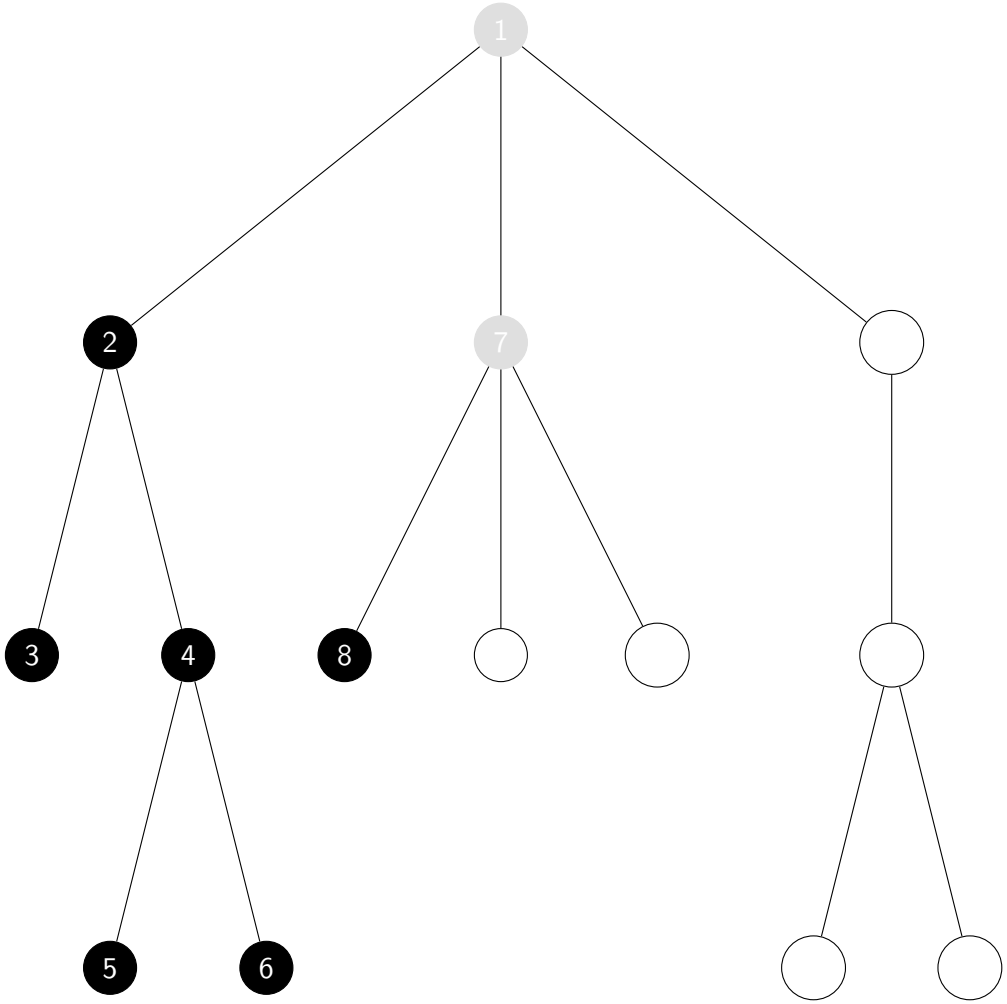
Parcours en profondeur d'un arbre



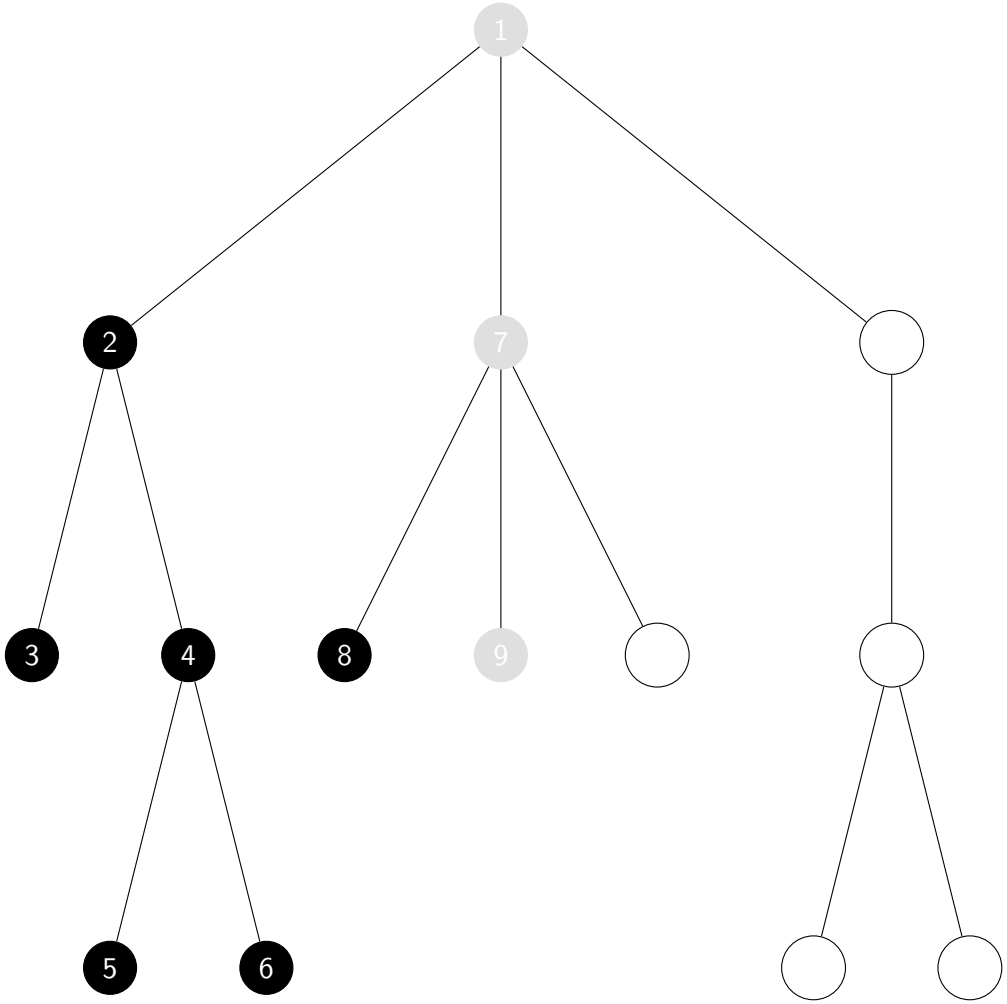
Parcours en profondeur d'un arbre



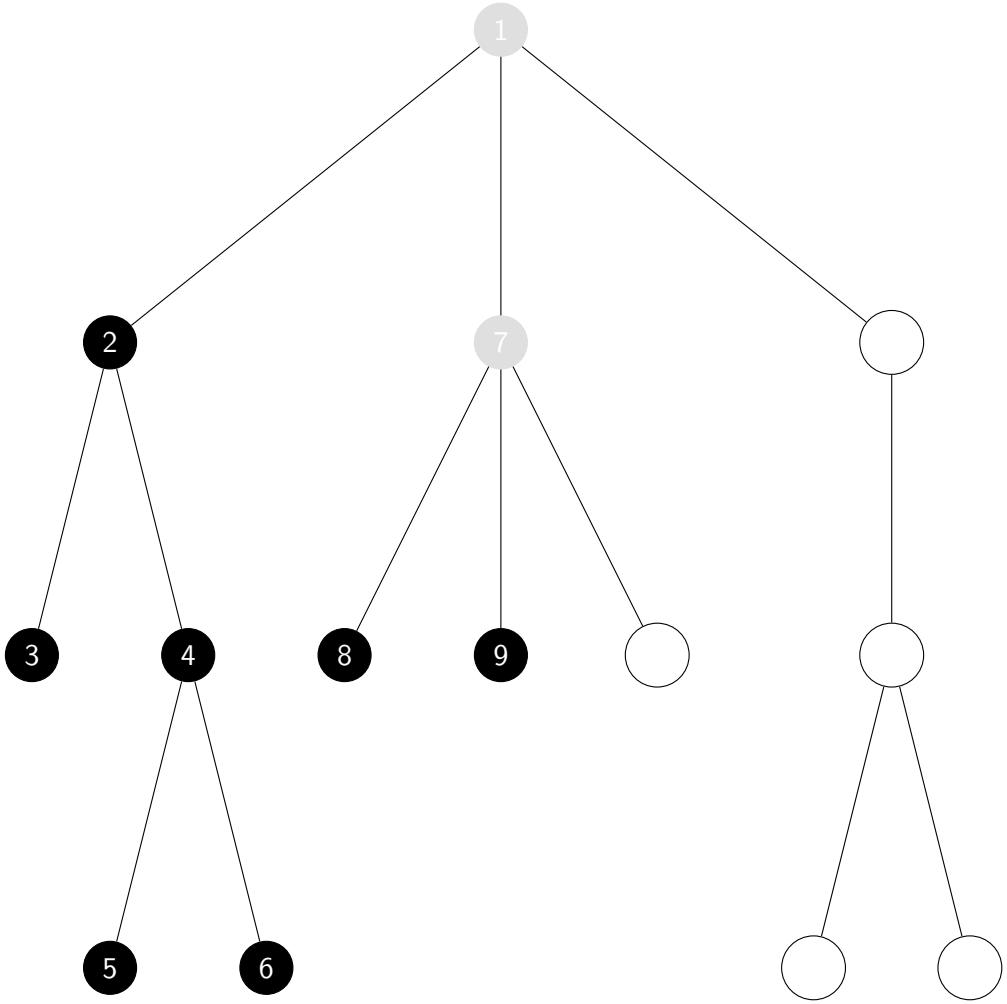
Parcours en profondeur d'un arbre



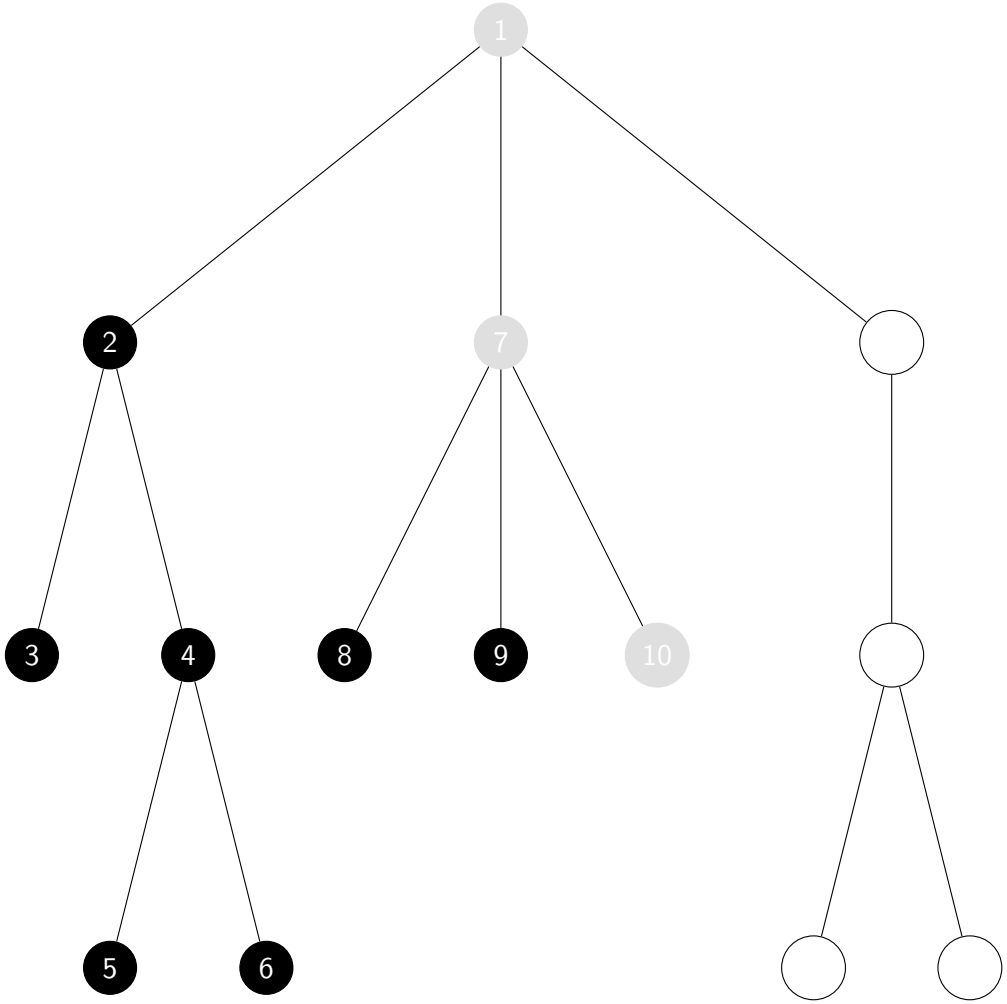
Parcours en profondeur d'un arbre



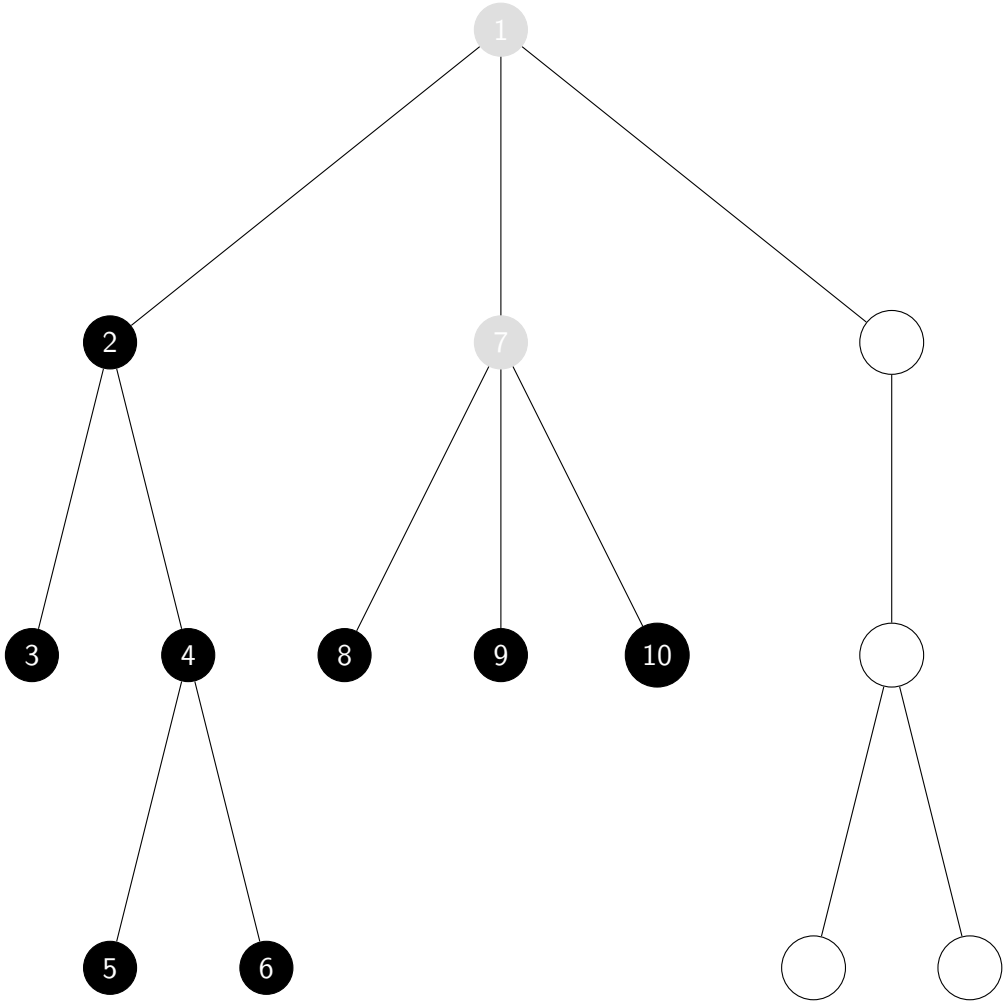
Parcours en profondeur d'un arbre



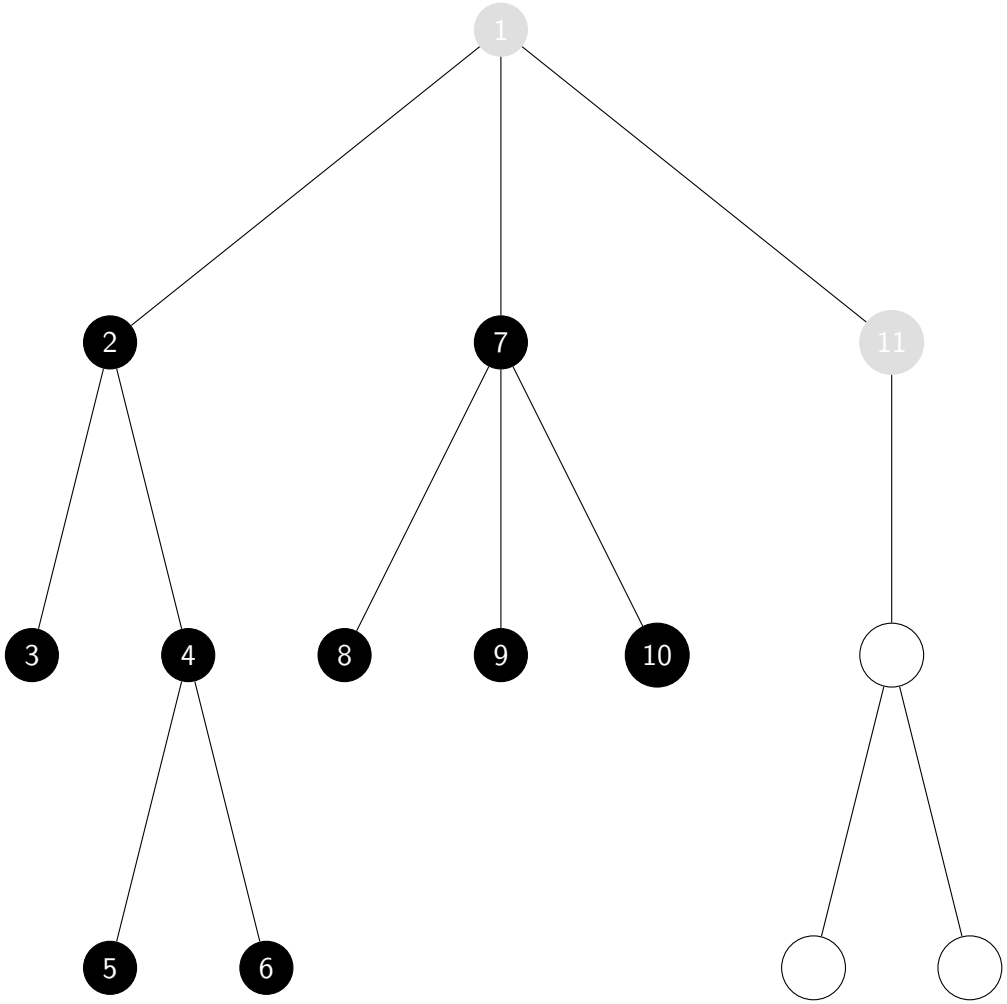
Parcours en profondeur d'un arbre



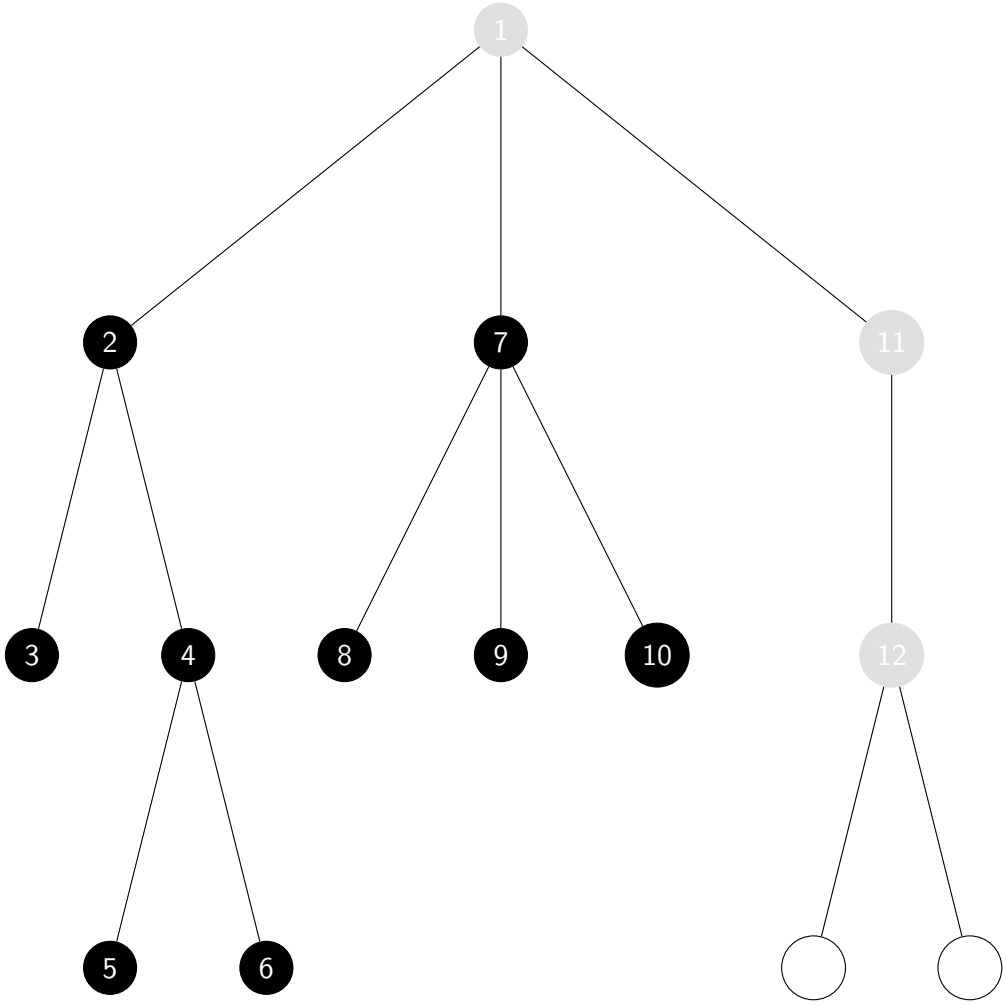
Parcours en profondeur d'un arbre



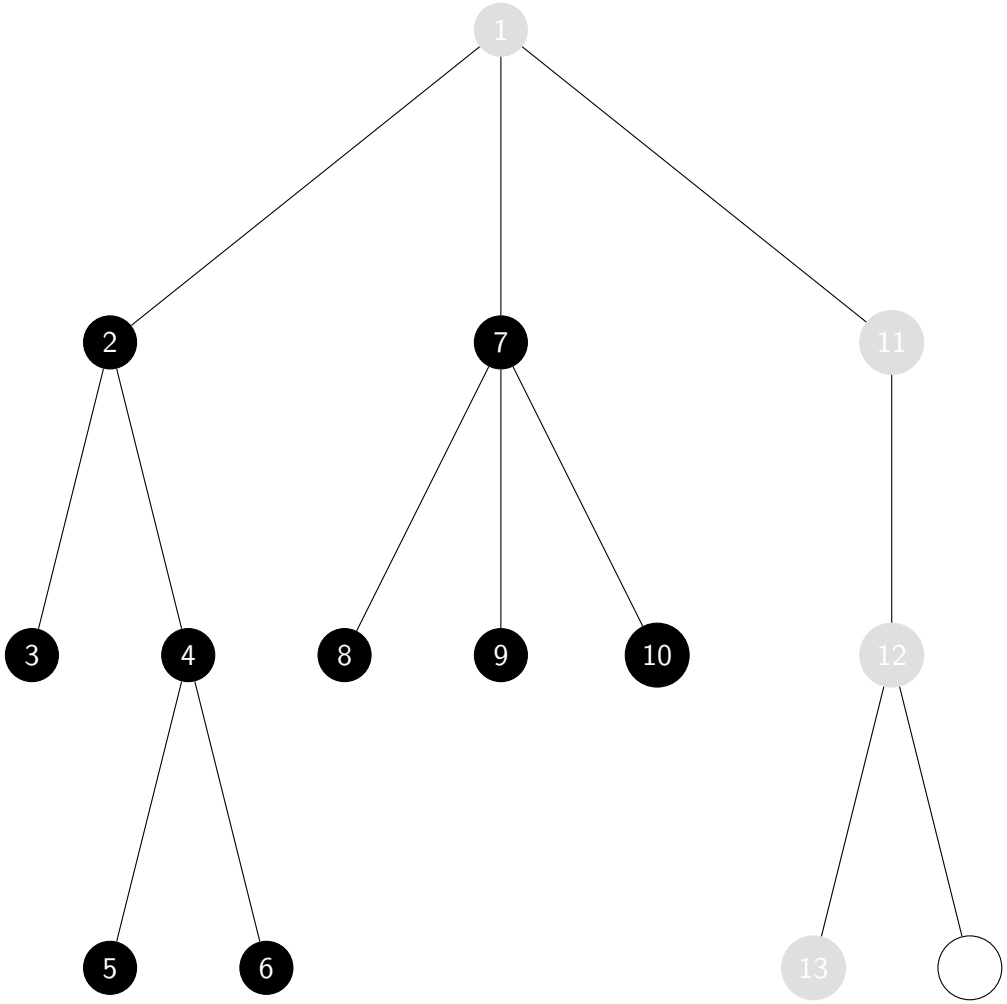
Parcours en profondeur d'un arbre



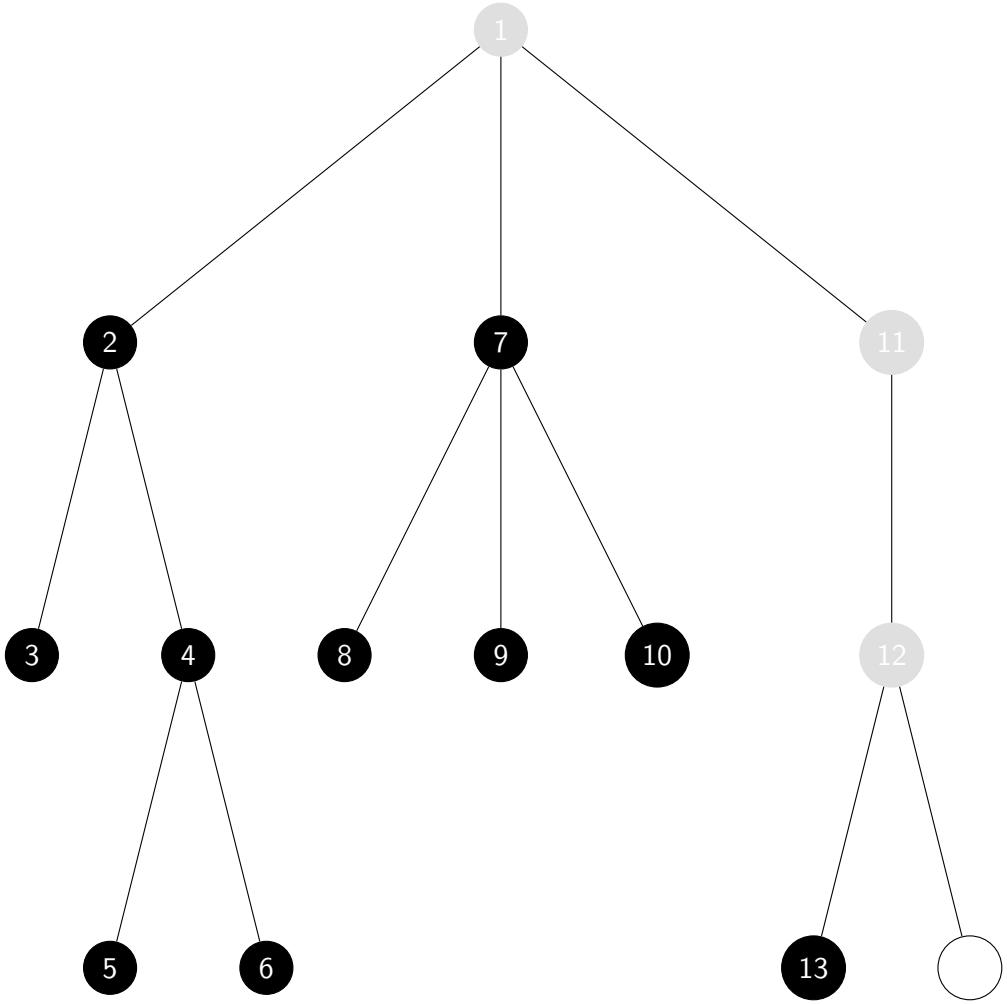
Parcours en profondeur d'un arbre



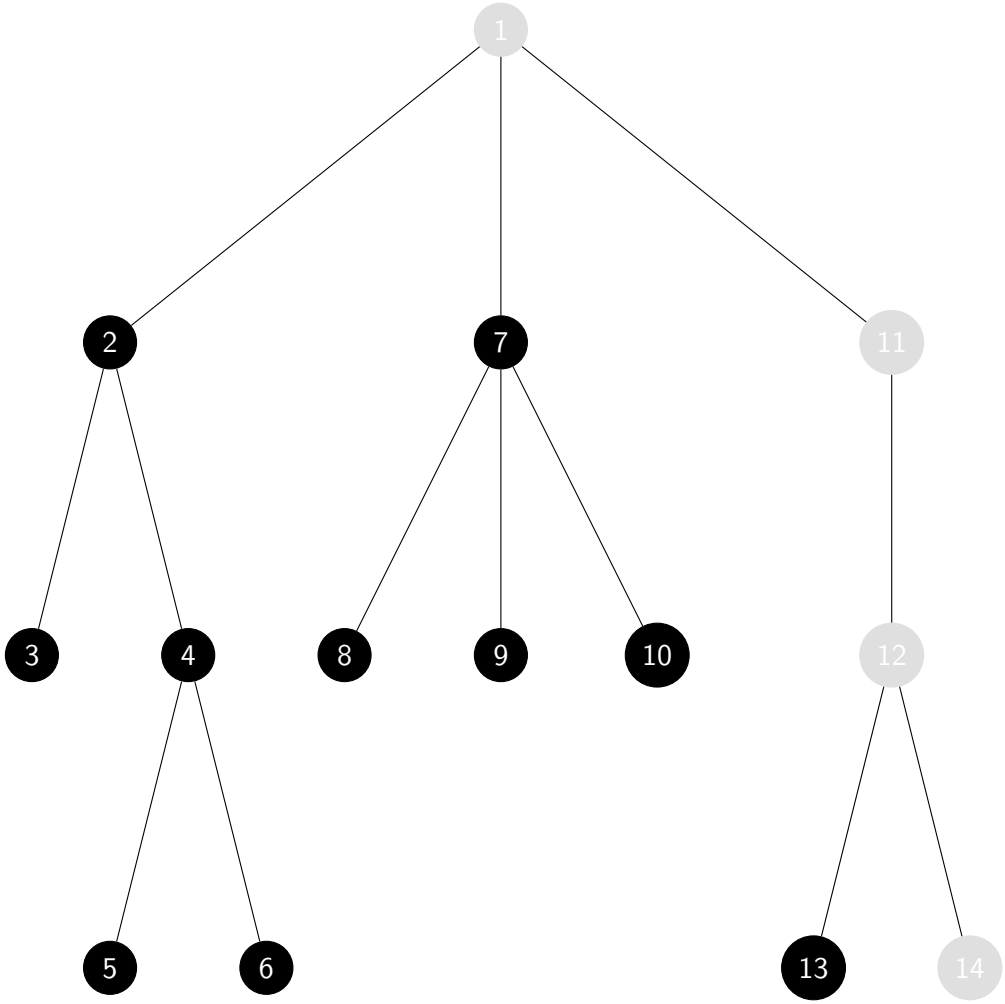
Parcours en profondeur d'un arbre



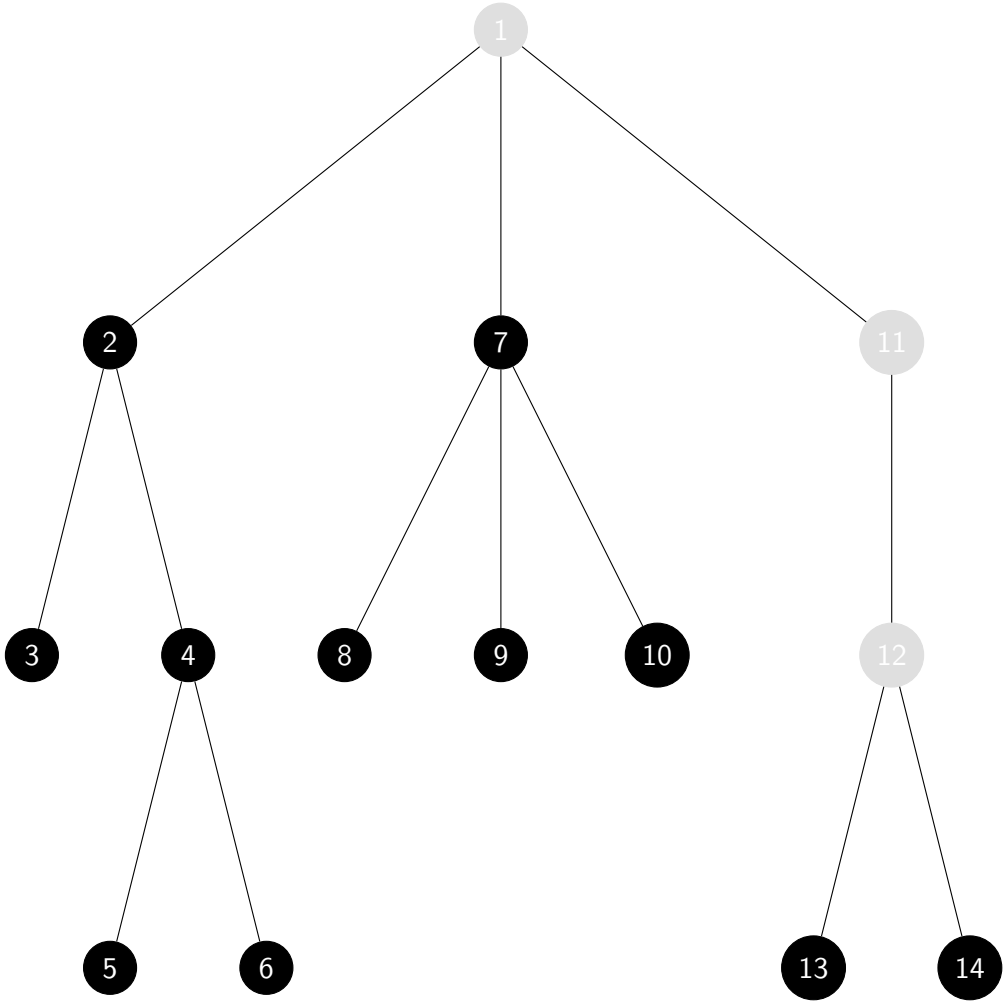
Parcours en profondeur d'un arbre



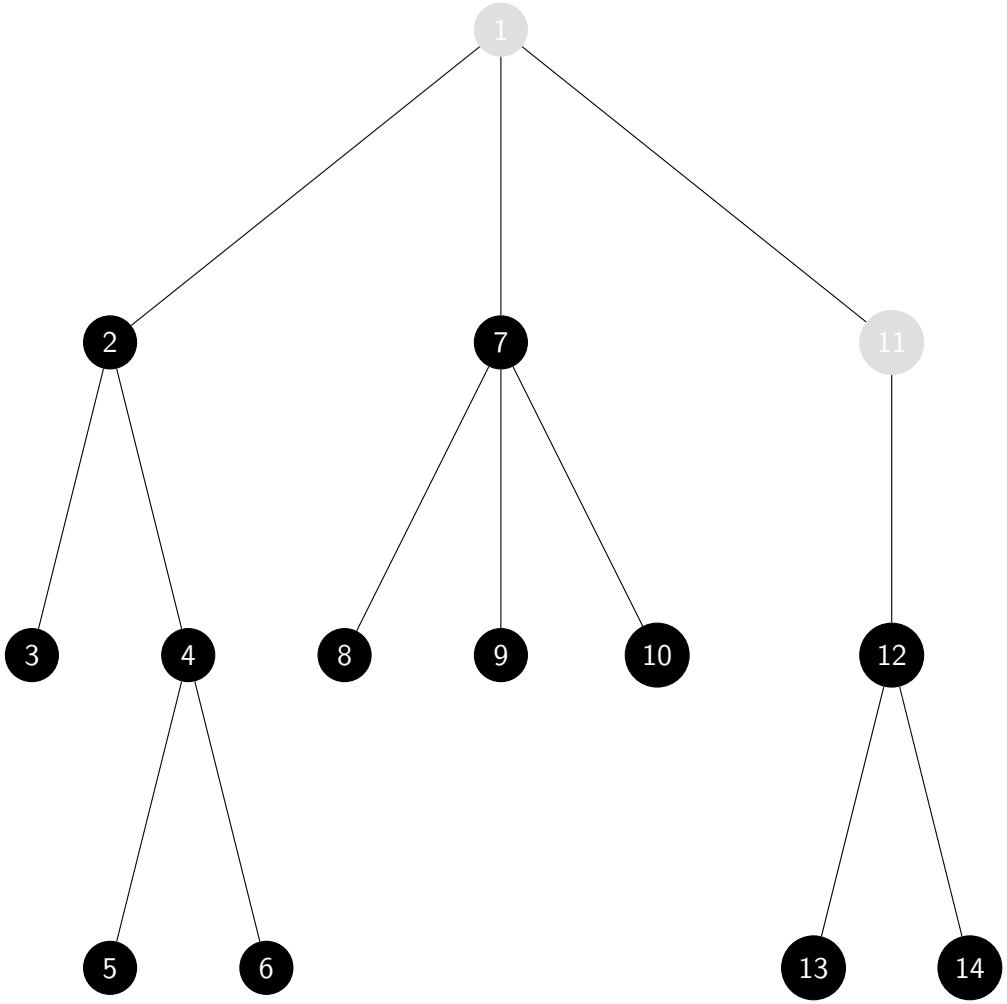
Parcours en profondeur d'un arbre



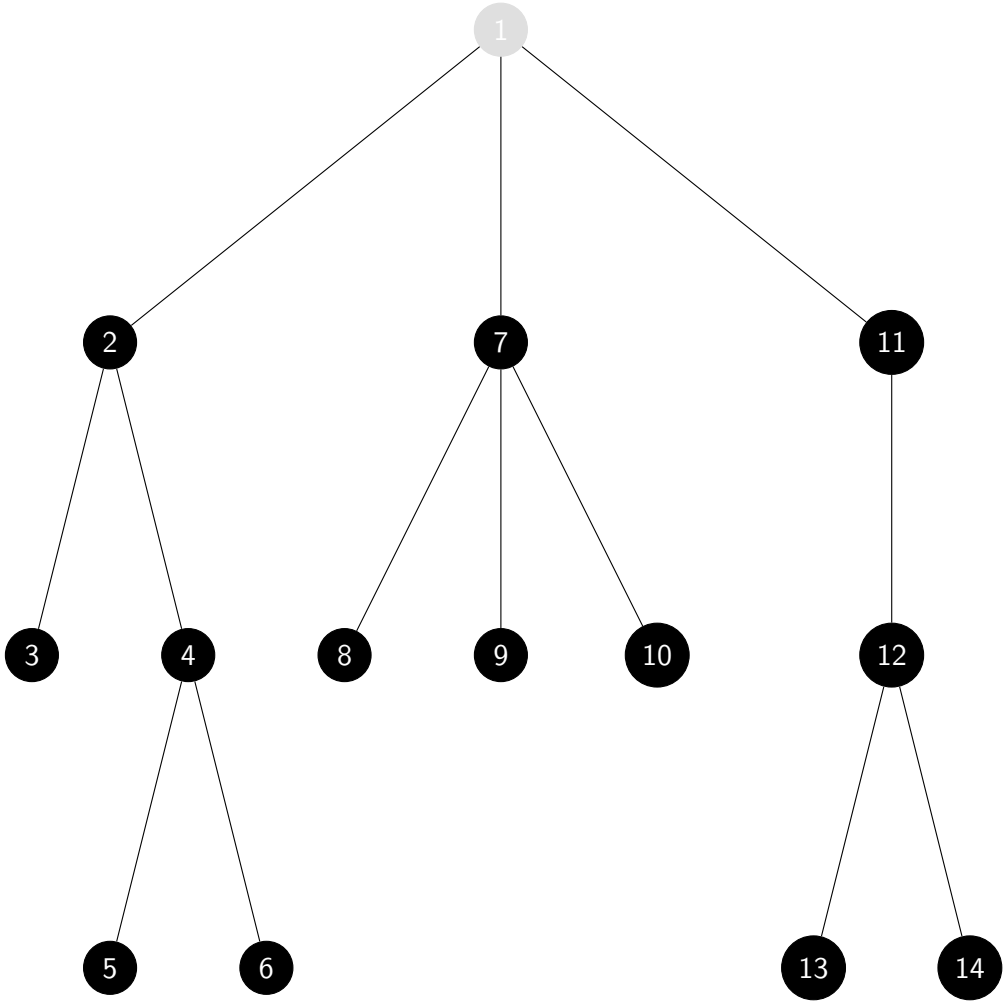
Parcours en profondeur d'un arbre



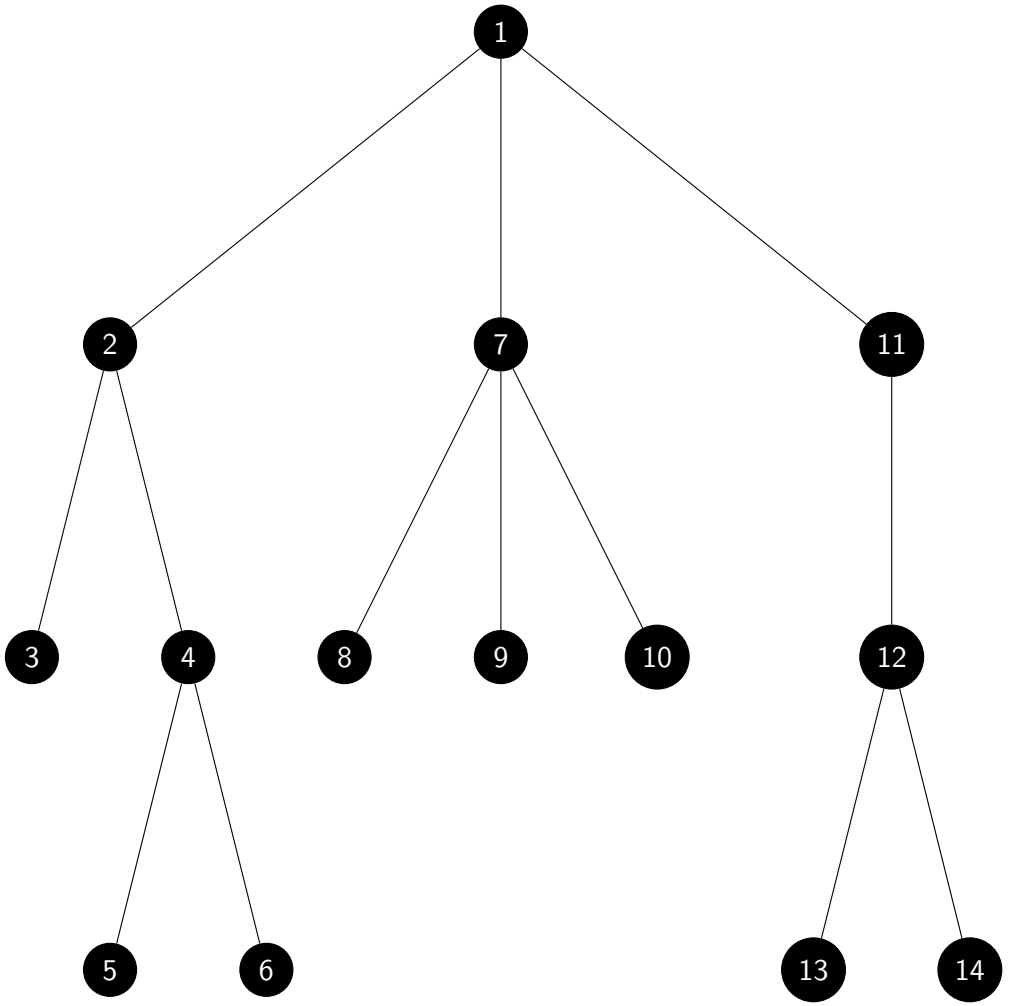
Parcours en profondeur d'un arbre



Parcours en profondeur d'un arbre



Parcours en profondeur d'un arbre



arbre DFS en python

DFS (depth first search) : programmation python

Exercice à rendre 3.

Avec la représentation d'un graphe par un dictionnaire comme précédemment, programmer en langage python le DFS avec les variables suivantes :

- Un dictionnaire P. En fin de parcours, pour tout sommet s du graphe P[s] sera le père de s, c'est à dire le sommet à partir duquel le sommet s a été découvert lors du parcours.

DFS (depth first search) : programmation python

Exercice à rendre 3.

Avec la représentation d'un graphe par un dictionnaire comme précédemment, programmer en langage python le DFS avec les variables suivantes :

- Un dictionnaire P. En fin de parcours, pour tout sommet s du graphe P[s] sera le père de s, c'est à dire le sommet à partir duquel le sommet s a été découvert lors du parcours.
- Un dictionnaire couleur. Pour tout sommet s, couleur[s] vaut blanc si le sommet s n'est pas encore découvert, gris s'il est déjà découvert mais non encore fermé (c'est à dire si l'algorithme n' a pas encore découvert tous ses voisins), noir lorsque ce sommet est fermé.

DFS (depth first search) : programmation python

Exercice à rendre 3.

Avec la représentation d'un graphe par un dictionnaire comme précédemment, programmer en langage python le DFS avec les variables suivantes :

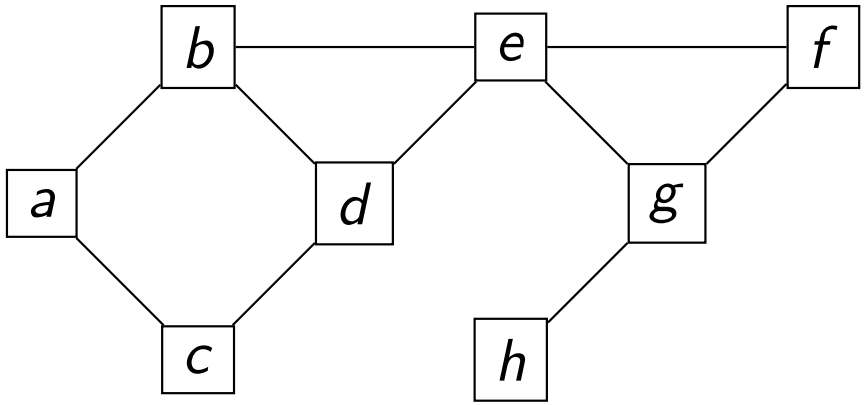
- Un dictionnaire P. En fin de parcours, pour tout sommet s du graphe P[s] sera le père de s, c'est à dire le sommet à partir duquel le sommet s a été découvert lors du parcours.
- Un dictionnaire couleur. Pour tout sommet s, couleur[s] vaut blanc si le sommet s n'est pas encore découvert, gris s'il est déjà découvert mais non encore fermé (c'est à dire si l'algorithme n' a pas encore découvert tous ses voisins), noir lorsque ce sommet est fermé.
- Une liste Q utilisée comme pile (lifo) : on empile un sommet lorsqu'il est découvert, on le dépile lorsqu'il est terminé (traitement prioritaire des sommets découverts au plus tard).

Exemple d'exécution

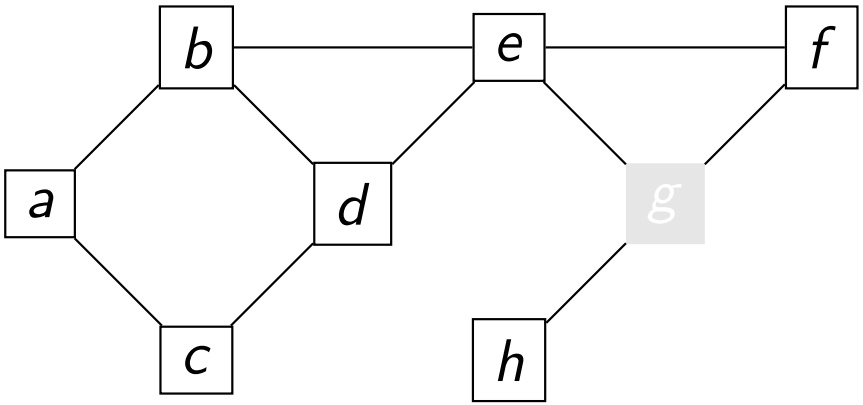
Déroulement attendu du programme, avec l'appel dfs(G,'g'), sur le graphe ci-dessous :

Python

```
G=dict()
G['a']=['b','c']
G['b']=['a','d','e']
G['c']=['a','d']
G['d']=['b','c','e']
G['e']=['b','d','f','g']
G['f']=['e','g']
G['g']=['e','f','h']
G['h']=['g']
```

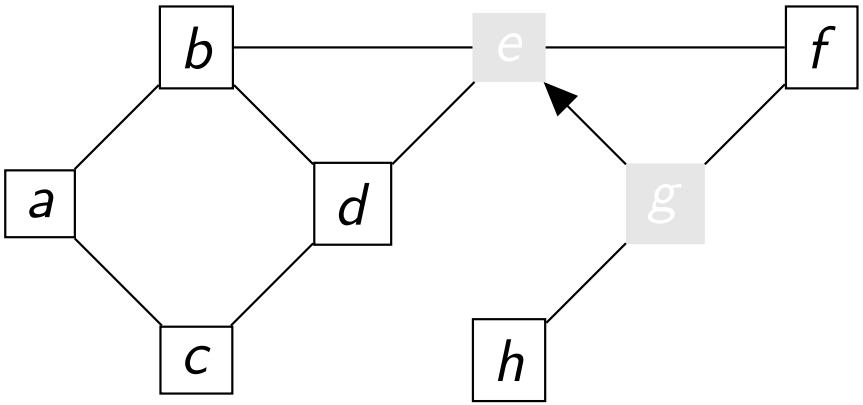


Déroulement



P={ g : None}
Q=[g]
Découverts (gris ou noirs)=[g]
Fermés (noirs) =[]

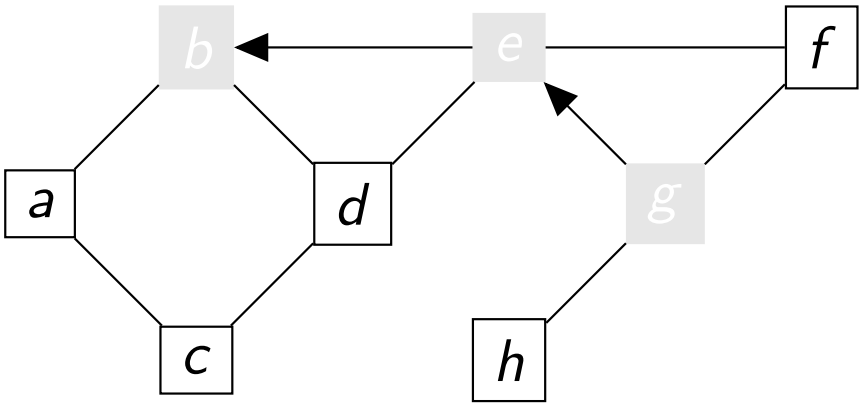
Déroulement



u=g, R=[e,f,h],v=e,

P={ g : None, e : g}
Q=[g,e]
Découverts=[g,e]
Fermés=[]

Déroulement



u=e, R=[b,d,f],v=b,

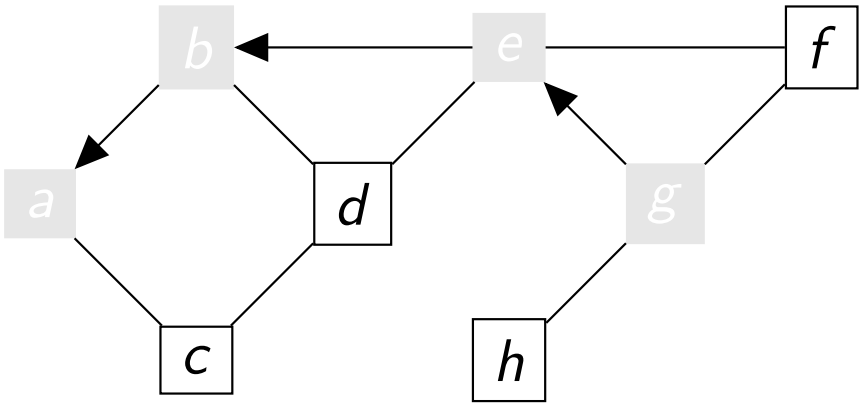
P={ g : None, e : g,b : e}

Q=[g,e,b]

Découverts=[g,e,b]

Fermés=[]

Déroulement



u=b, R=[a,d],v=a,

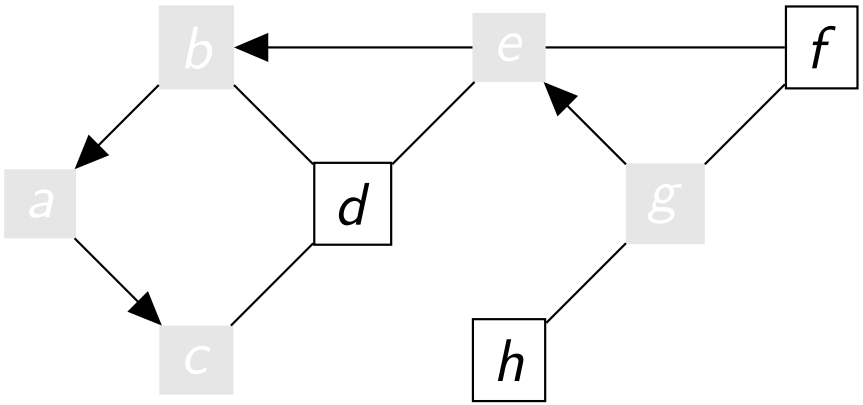
P={ g : None, e : g,b : e, a :b}

Q=[g,e,b,a]

Découverts=[g,e,b,a]

Fermés=[]

Déroulement



u=a, R=[c],v=c,

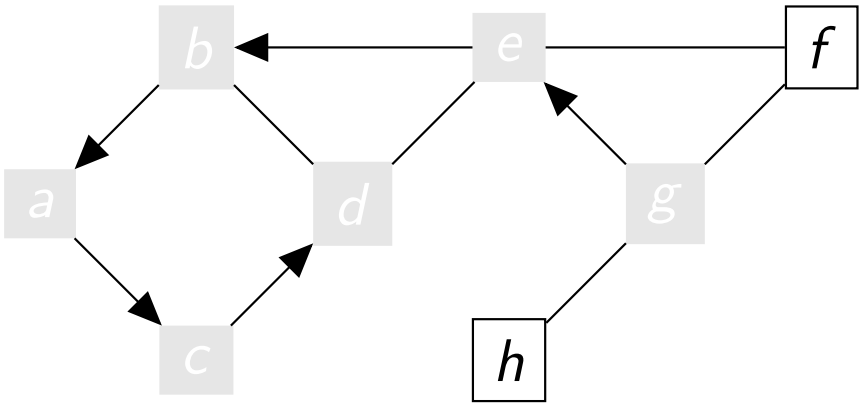
P={ g : None, e : g,b : e, a :b,c :a}

Q=[g,e,b,a,c]

Découverts=[g,e,b,a,c]

Fermés=[]

Déroulement



u=c, R=[d],v=d,

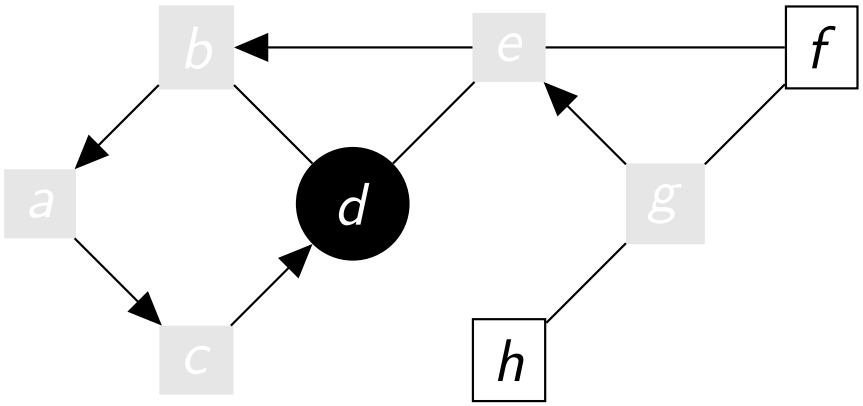
P={ g : None, e : g,b : e, a :b,c :a, d :c}

Q=[g,e,b,a,c,d]

Découverts=[g,e,b,a,c,d]

Fermés=[]

Déroulement



u=d, R=[],

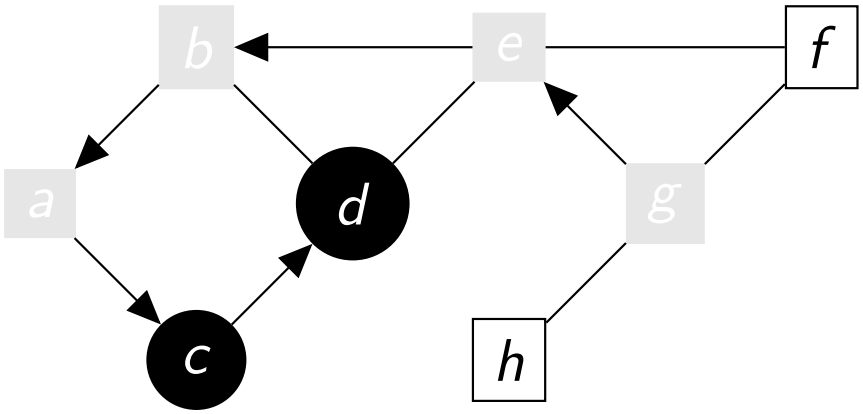
P={ g : None, e : g,b : e, a :b,c :a, d :c}

Q=[g,e,b,a,c]

Découverts=[g,e,b,a,c,d]

Fermés=[d]

Déroulement



u=c, R=[],

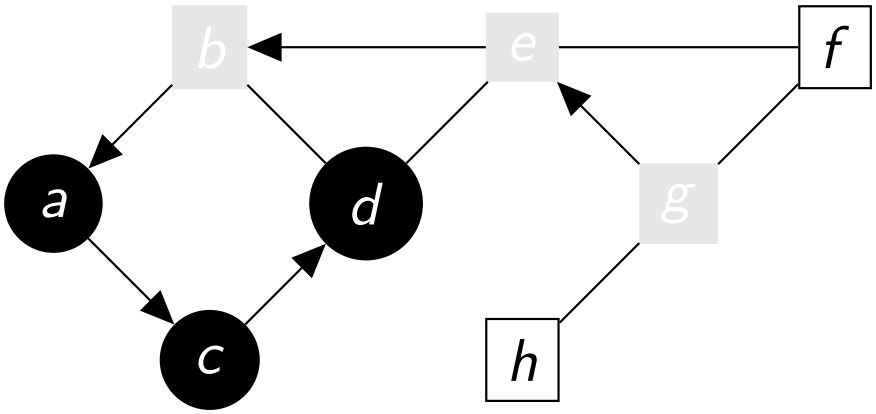
P={ g : None, e : g,b : e, a :b,c :a, d :c}

Q=[g,e,b,a]

Découverts=[g,e,b,a,c,d]

Fermés=[d,c]

Déroulement



u=a, R=[],

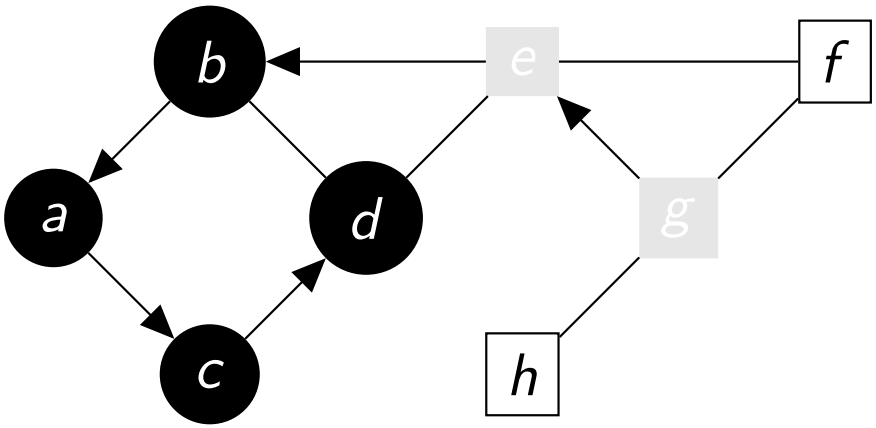
P={ g : None, e : g,b : e, a :b,c :a, d :c}

Q=[g,e,b]

Découverts=[g,e,b,a,c,d]

Fermés=[d,c,a]

Déroulement



u=b, R=[],

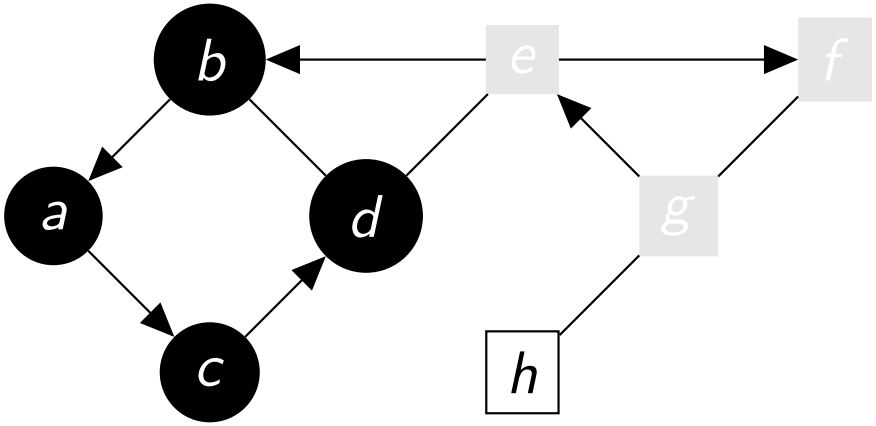
P={ g : None, e : g,b : e, a :b,c :a, d :c}

Q=[g,e]

Découverts=[g,e,b,a,c,d]

Fermés=[d,c,a,b]

Déroulement



u=e, R=[f], v=f,

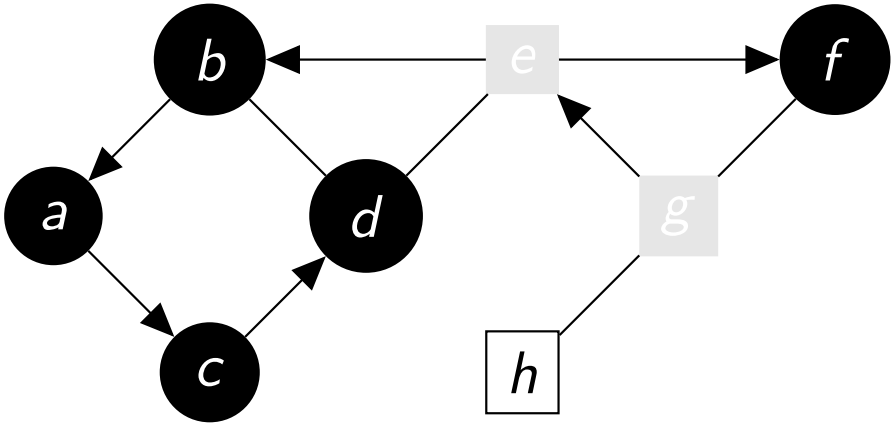
P={ g : None, e : g,b : e, a :b,c :a, d :c,f :e}

Q=[g,e,f]

Découverts=[g,e,b,a,c,d,f]

Fermés=[d,c,a,b]

Déroulement



u=f, R=[],

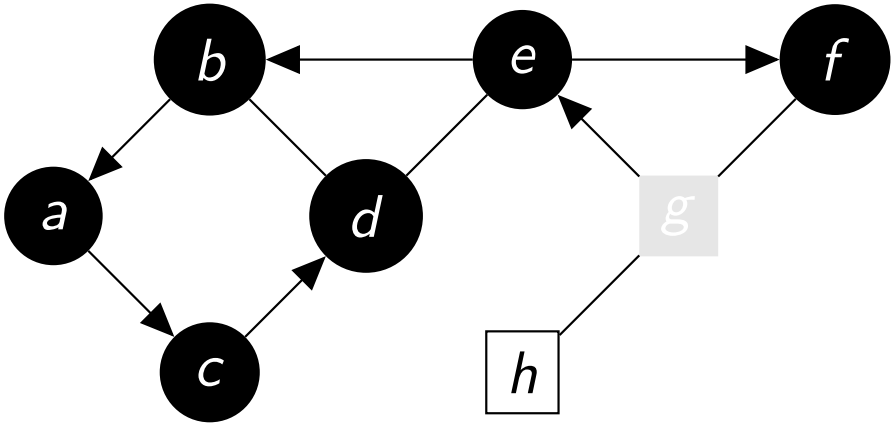
P={ g : None, e : g,b : e, a :b,c :a, d :c,f :e}

Q=[g,e]

Découverts=[g,e,b,a,c,d,f]

Fermés=[d,c,a,b,f]

Déroulement



u=e, R=[],

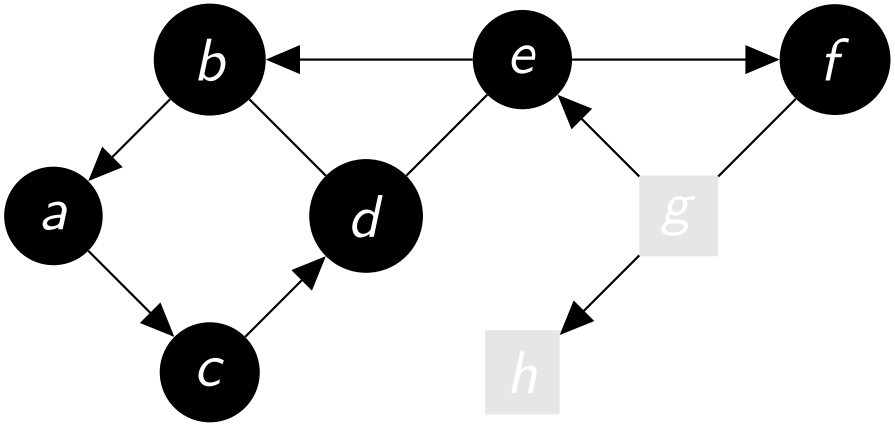
P={ g : None, e : g,b : e, a :b,c :a, d :c,f :e}

Q=[g]

Découverts=[g,e,b,a,c,d,f]

Fermés=[d,c,a,b,f,e]

Déroulement



u=g, R=[h], v=h,

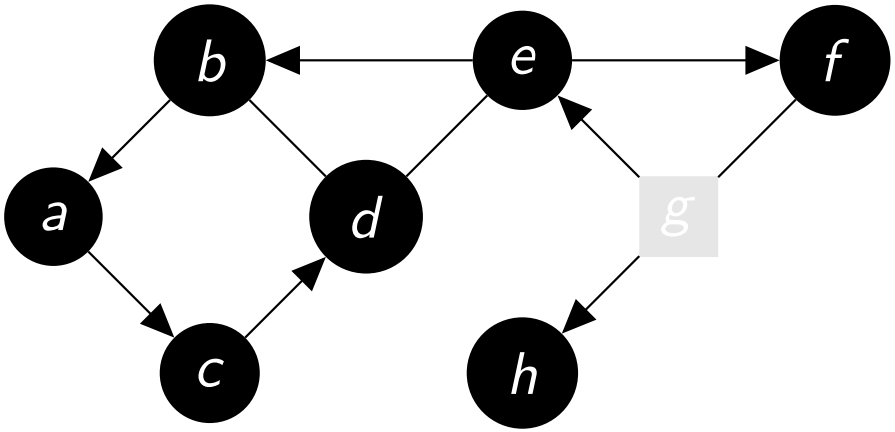
P={ g : None, e : g,b : e, a :b,c :a, d :c,f :e,h :g}

Q=[g,h]

Découverts=[g,e,b,a,c,d,f,h]

Fermés=[d,c,a,b,f,e]

Déroulement



u=h, R=[],

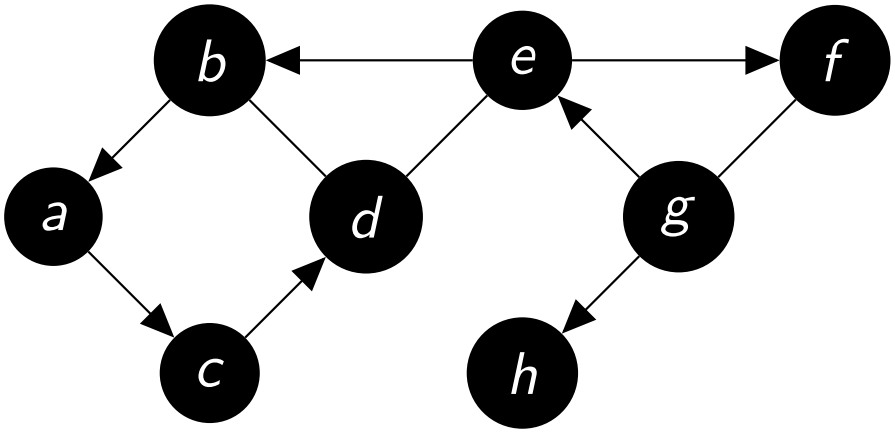
P={ g : None, e : g,b : e, a :b,c :a, d :c,f :e,h :g}

Q=[g]

Découverts=[g,e,b,a,c,d,f,h]

Fermés=[d,c,a,b,f,e,h]

Déroulement



u=g, R=[],

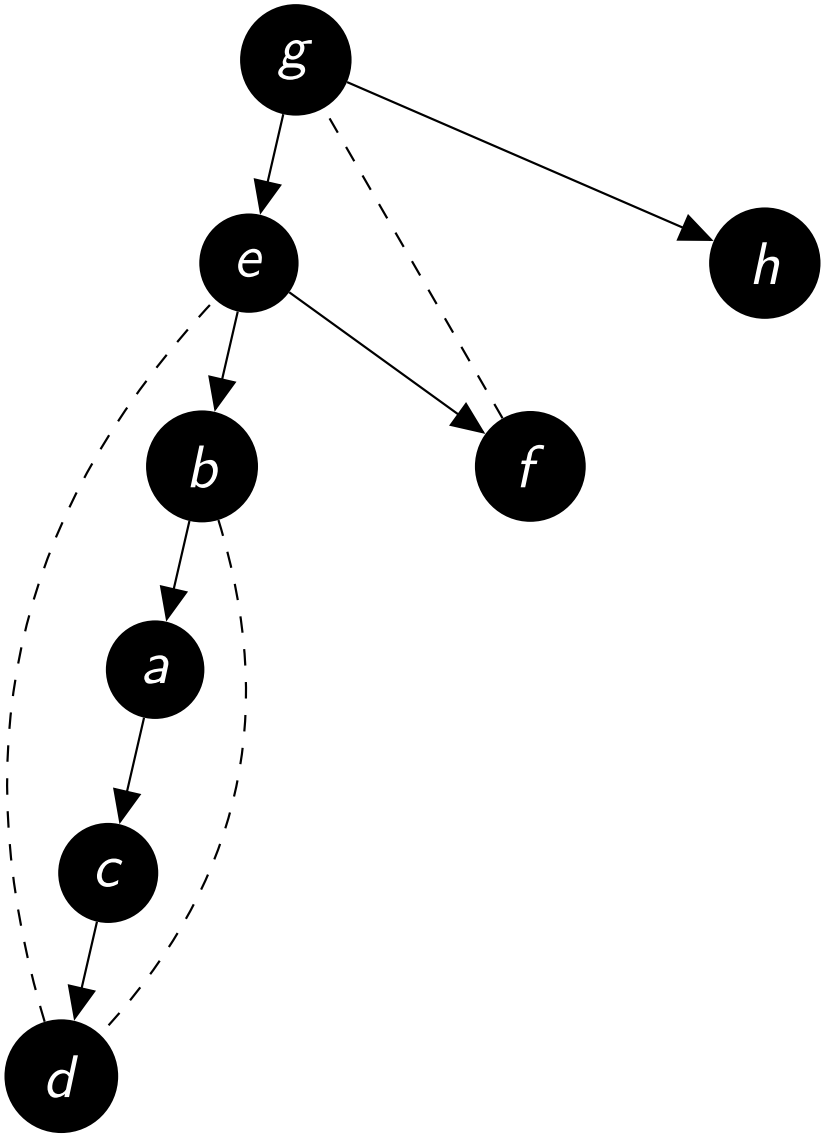
P={ g : None, e : g,b : e, a :b,c :a, d :c,f :e,h :g}

Q=[ ]

Découverts=[g,e,b,a,c,d,f,h]

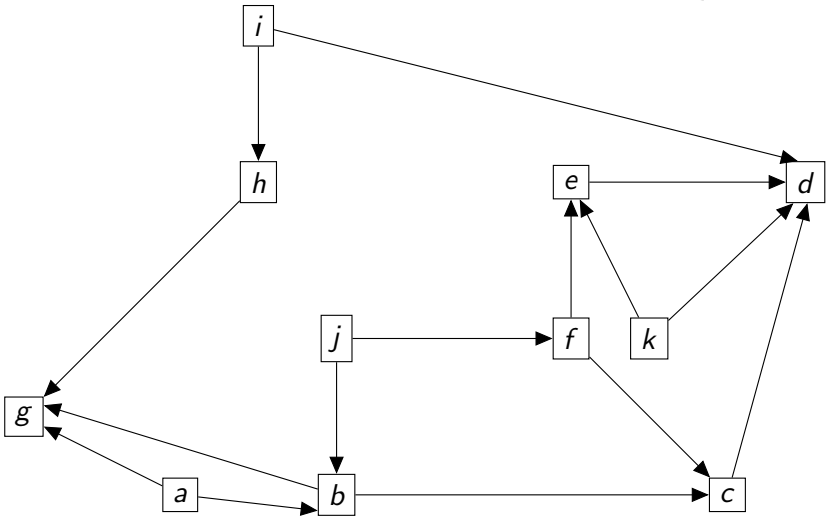
Fermés=[d,c,a,b,f,e,h,g]

Arborescence associée au parcours



tri topologique

Exemple d'application du parcours en profondeur.
Début d'année, vos enseignants préparent le cours d'ISN.
Pour cela, ils découpent les notions en chapitres a, b, c, d...
Sur le graphe ci-dessous, une flèche du chapitre x vers le chapitre y signifie que x doit être traité avant y.



Pour une meilleure organisation et savoir où commencer , on aimerait réorganiser la représentation du graphe de façon à ce que tous les sommets soient dessinés alignés, toutes les flèches du schéma devant être orientées vers la droite.

tri topologique

L'énumération des sommets du graphe en ordre inverse de leur date de fermeture dans l'algorithme de parcours en profondeur permet d'obtenir un ordre des sommets satisfaisant la demande.

Python

```
def lancement(G) :
    for s in G :
        if couleur[s]=='blanc' : parcours(G,s)
def parcours(G,s) :
    couleur[s]='gris'
    for v in G[s] :
        if couleur[v]=='blanc' : parcours(G,v)
    P.append(s)
# # # # # #
couleur=dict()
for v in G :couleur[v]='blanc'
P=list()
lancement(G)
P.reverse()
print P
```

tri topo python