Parcours dans les graphes

Graphes 2019

Principe

Le principe est de parcourir le graphe suivant une méthode donnée. Tout au long du parcours les sommets peuvent prendre trois états :

1. non marqué, tant que le sommet n'a pas été "visité",

Principe

Le principe est de parcourir le graphe suivant une méthode donnée. Tout au long du parcours les sommets peuvent prendre trois états :

1. non marqué, tant que le sommet n'a pas été "visité",
2. ouvert lors de la première visite du sommet

Principe

Le principe est de parcourir le graphe suivant une méthode donnée. Tout au long du parcours les sommets peuvent prendre trois états :

1. non marqué, tant que le sommet n'a pas été "visité",
2. ouvert lors de la première visite du sommet
3. marqué ou fermé lorsque le sommet a été visité.

.

Principe

Le principe est de parcourir le graphe suivant une méthode donnée. Tout au long du parcours les sommets peuvent prendre trois états :

1. non marqué, tant que le sommet n'a pas été "visité",
2. ouvert lors de la première visite du sommet
3. marqué ou fermé lorsque le sommet a été visité.

Au début, tous les sommets sont non marqués. Il faut alors choisir un premier sommet à ouvrir qu'on appelle sommet initial. Puis à chaque étape un choix est effectué pour l'ordre d'ouverture (**ordre de prévisite)** et de fermeture des sommets (**ordre de postvisite)** .

Parcours générique dans un graphe orienté

Algorithme 1 Algorithme de parcours générique pour graphe orienté

Entrées: G : *graphe*

Initialement tous les sommets sont non marqués
tantque il existe un sommet s non marqué **faire**
    ouvrir s
    tantque cela est possible **faire**
        ouvrir un sommet y non marqué s'il est successeur à un sommet ouvert x
        fermer un sommet x si tous ses sommets successeurs sont ouverts ou fermés
    fin tantque
fin tantque

Exemple

PILE et FILE

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 : principe de l'algorithme

BFS (breadth first search)

Parcours en largeur : principe de l'algorithme

BFS (breadth first search)

1. On utilise une file. On enfile le sommet de départ.

Parcours en largeur : principe de l'algorithme

BFS (breadth first search)

1. On utilise une file. On enfile le sommet de départ.
2. On visite les voisins de la 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

BFS (breadth first search)

1. On utilise une file. On enfile le sommet de départ.
2. On visite les voisins de la 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).

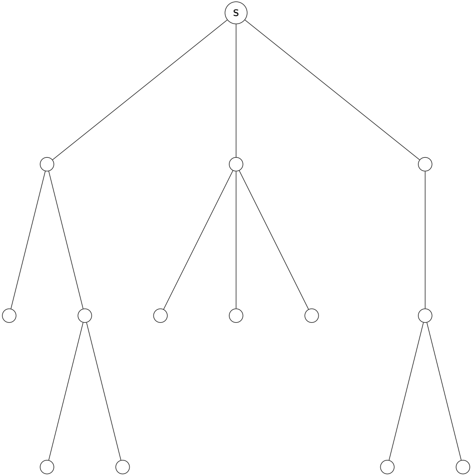
Parcours en largeur : principe de l'algorithme

BFS (breadth first search)

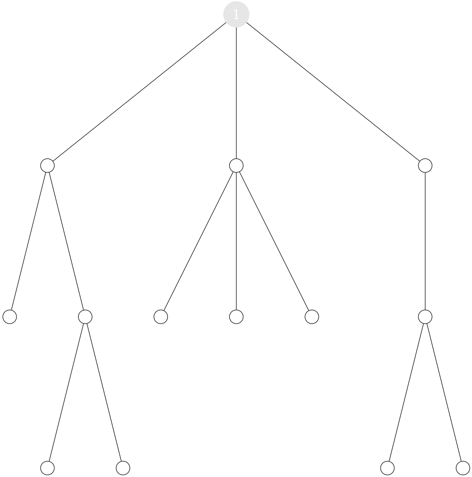
1. On utilise une file. On enfile le sommet de départ.
2. On visite les voisins de la tête de file. On les enfile (en les numérotant au fur et à mesure de leur découverte) s'ils ne sont pas déjà présents dans la file, ni déjà passés dans la file.
3. On défile (c'est à dire : on supprime la tête de file).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

Parcours en largeur d'un arbre

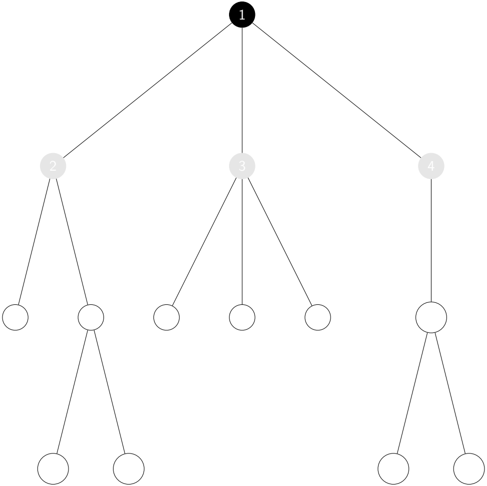
Parcourir en largeur le graphe ci-dessous à partir du sommet s :



Parcours en largeur d'un arbre



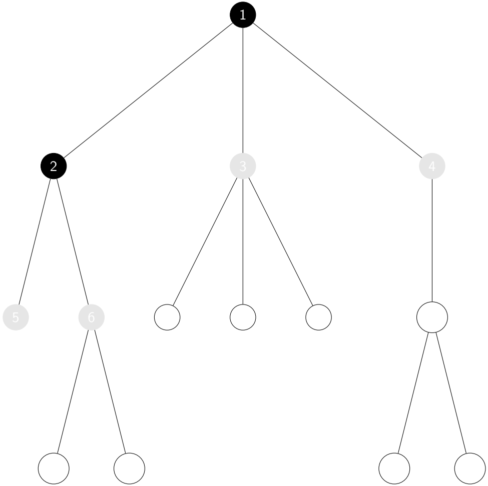
Parcours en largeur d'un arbre



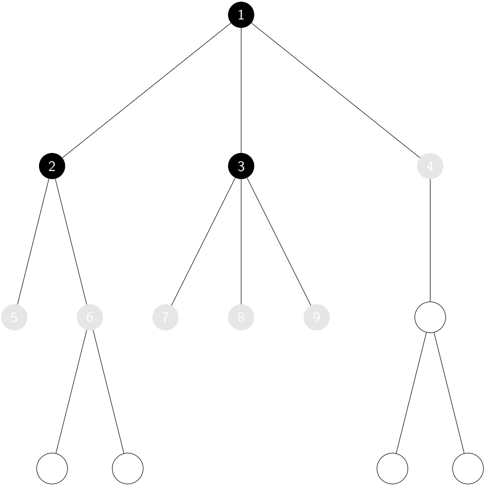
Enfiler : passage en gris. Défiler : passage en noir.

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

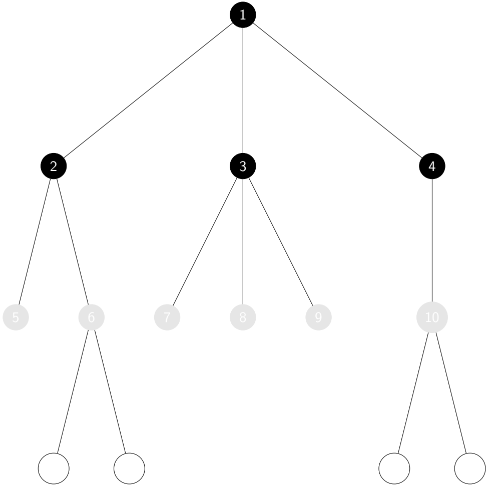
Parcours en largeur d'un arbre



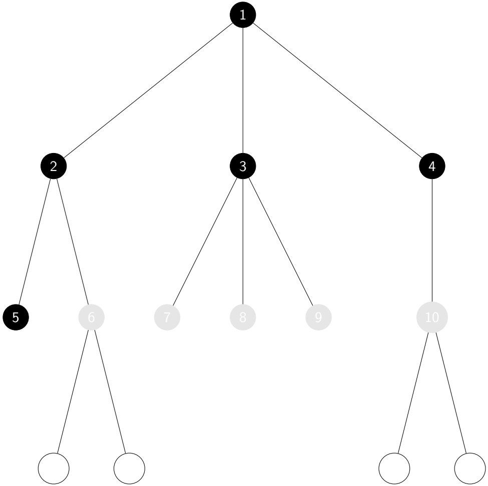
Parcours en largeur d'un arbre



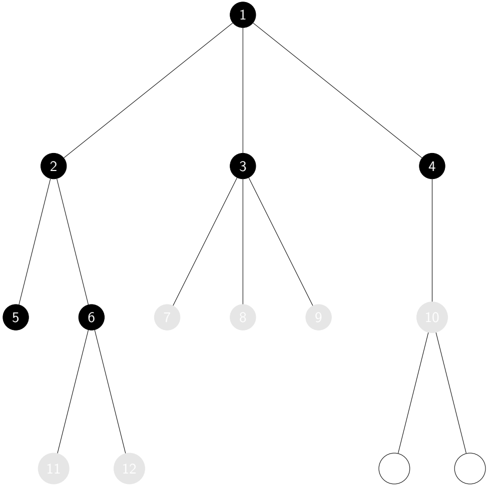
Parcours en largeur d'un arbre



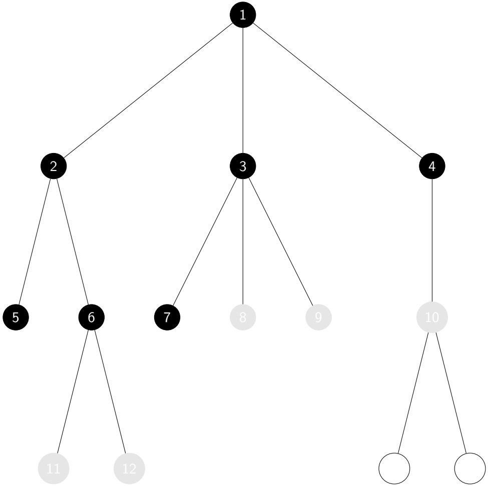
Parcours en largeur d'un arbre



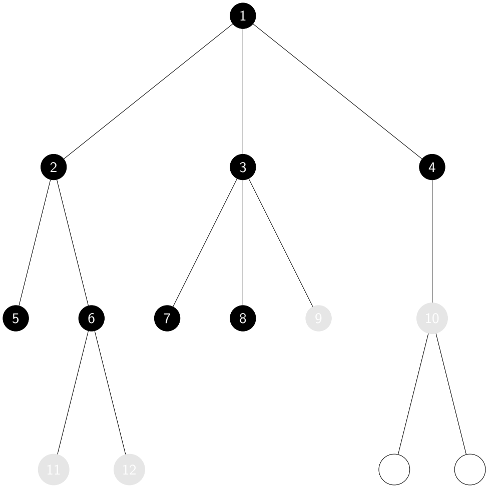
Parcours en largeur d'un arbre



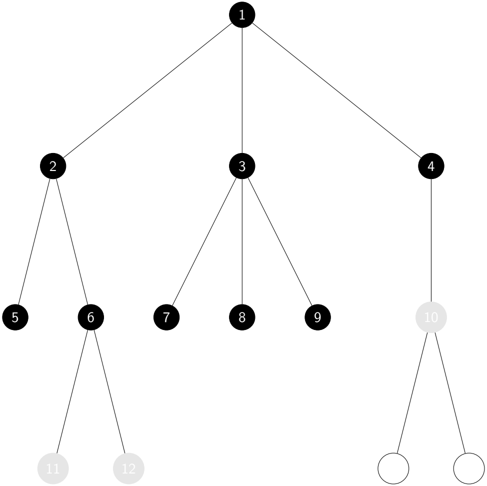
Parcours en largeur d'un arbre



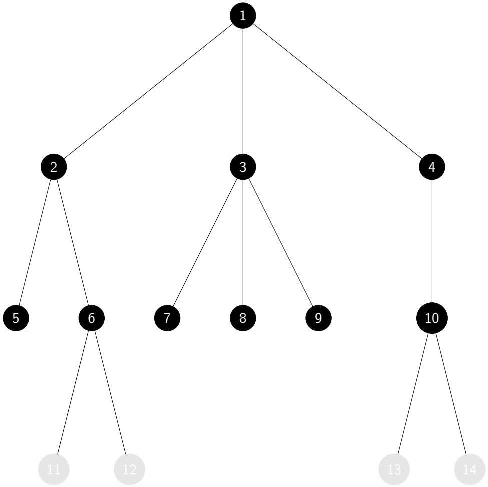
Parcours en largeur d'un arbre



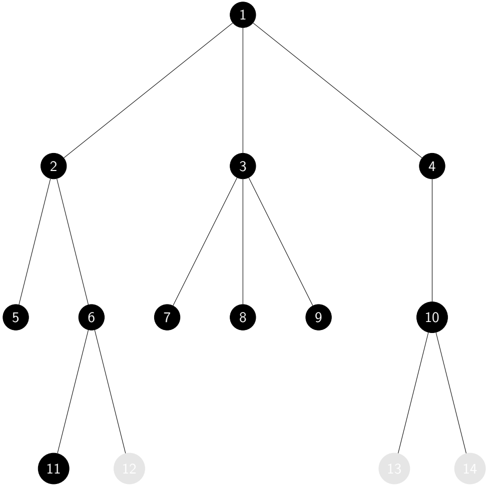
Parcours en largeur d'un arbre



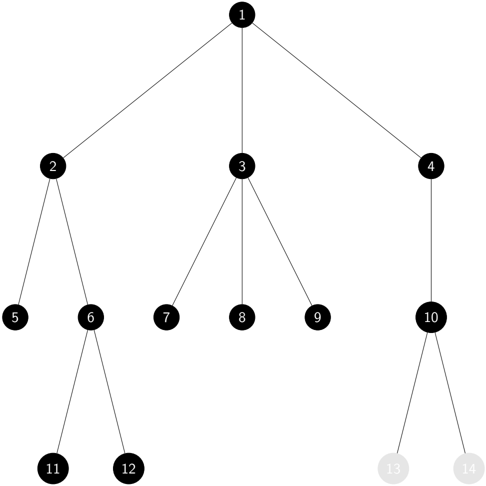
Parcours en largeur d'un arbre



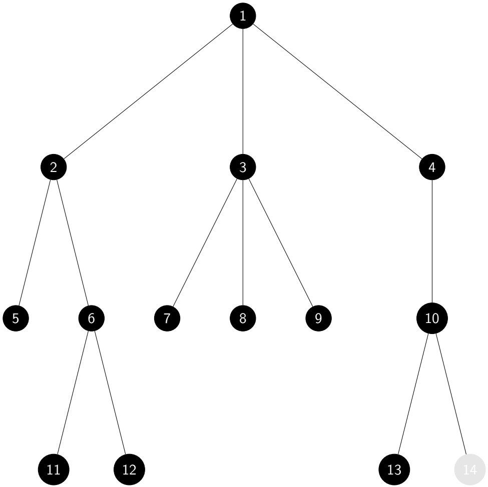
Parcours en largeur d'un arbre



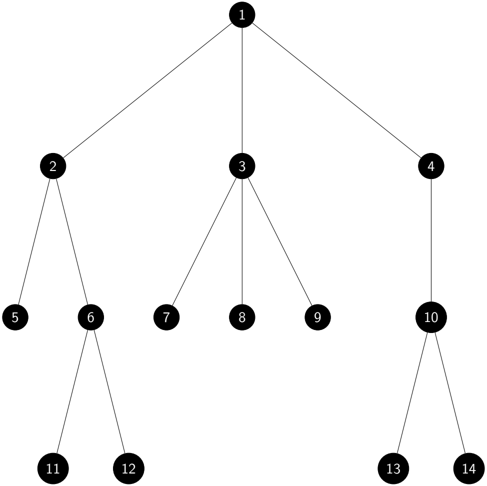
Parcours en largeur d'un arbre



Parcours en largeur d'un arbre

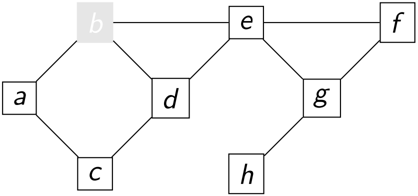


Parcours en largeur d'un arbre



L'algorithme de parcours en largeur va visiter en premier lieu toutes les noeuds à distance 1 du départ, puis toutes les noeuds à distance 2 du départ, puis toutes les noeuds à distance 3. . .

Déroulement



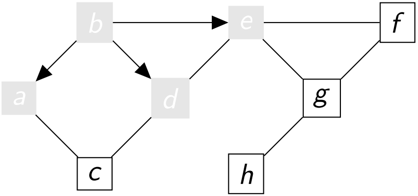
P={ 'b' : None}

Q=['b']

Découverts (gris ou noirs) =['b']

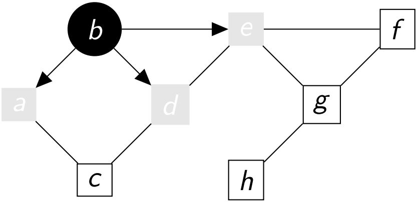
Fermés (noirs) =[]

Déroulement



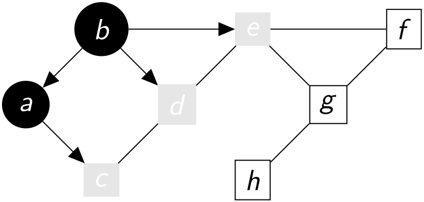
P={ 'b' : None, 'a' :'b', 'd' :'b', 'e' :'b'}
Q=['b','a','d','e']
Découverts =['b','a','d','e']
Fermés=[]

Déroulement



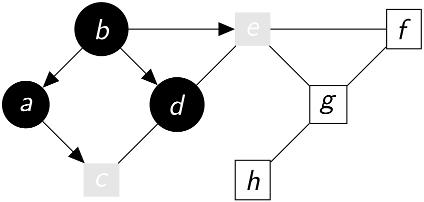
P={ 'b' : None, 'a' :'b', 'd' :'b', 'e' :'b'}
Q=['a','d','e']
Découverts=['b','a','d','e']
Fermés=['b']

Déroulement



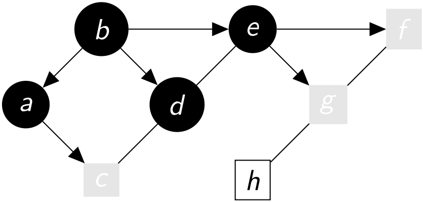
P={ 'b' : None, 'a' :'b', 'd' :'b', 'e' :'b','c' :'a'}
Q=['d','e','c']
Découverts=['b','a','d','e','c']
Fermés=['b','a']

Déroulement



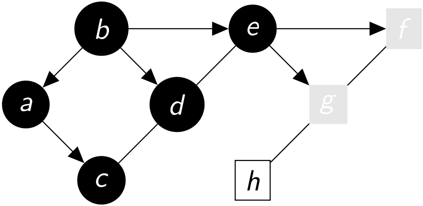
P={ 'b' : None, 'a' :'b', 'd' :'b', 'e' :'b','c' :'a'}
Q=['e','c']
Découverts=['b','a','d','e','c']
Fermés=['b','a','d']

Déroulement



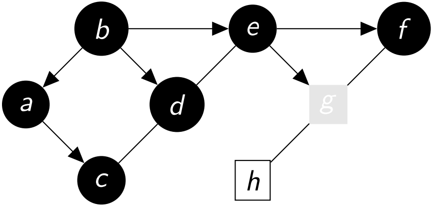
P={ 'b' : None, 'a' :'b', 'd' :'b', 'e' :'b','c' :'a','f' :'e','g' :'e'}
Q=['c','f','g']
Découverts=['b','a','d','e','c','f','g']
Fermés=['b','a','d','e']

Déroulement



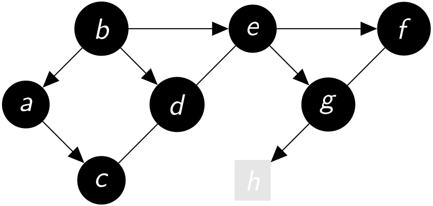
P={ 'b' : None, 'a' :'b', 'd' :'b', 'e' :'b','c' :'a','f' :'e','g' :'e'}
Q=['f','g']
Découverts=['b','a','d','e','c','f','g']
Fermés=['b','a','d','e','c']

Déroulement



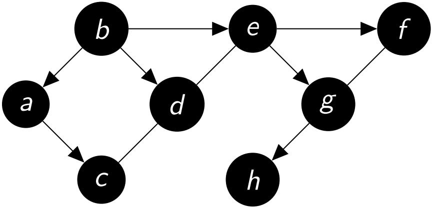
P={ 'b' : None, 'a' :'b', 'd' :'b', 'e' :'b','c' :'a','f' :'e','g' :'e'}
Q=['g']
Découverts=['b','a','d','e','c','f','g']
Fermés=['b','a','d','e','c','f']

Déroulement



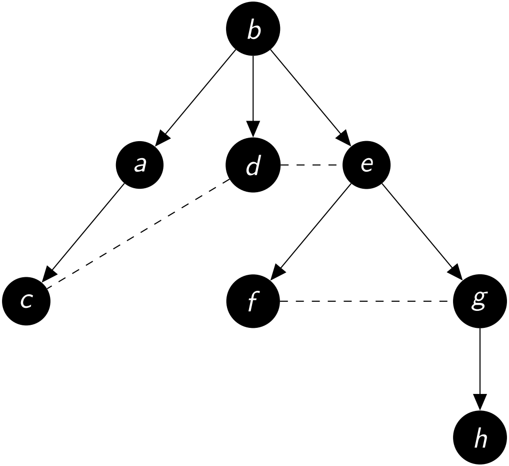
P={ 'b' : None, 'a' :'b', 'd' :'b', 'e' :'b','c' :'a','f' :'e','g' :'e','h' :'g'}
Q=['h']
Découverts=['b','a','d','e','c','f','g','h']
Fermés=['b','a','d','e','c','f','g']

Déroulement

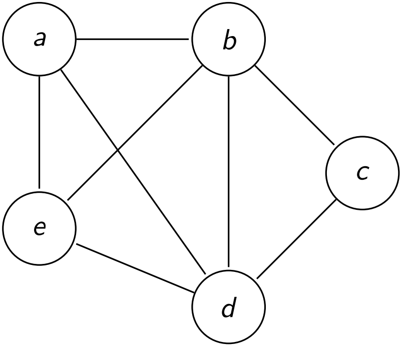


P={ 'b' : None, 'a' :'b', 'd' :'b', 'e' :'b','c' :'a','f' :'e','g' :'e','h' :'g'}
Q=[]
Découverts=['b','a','d','e','c','f','g','h']
Fermés=['b','a','d','e','c','f','g','h']

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.


a
b
c
d
e

En 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
```

DFS (Depth first search)

Parcours en profondeur

Parcours en profondeur : principe de l'algorithme

Parcours en profondeur : principe de l'algorithme

1. Dans le parcours en profondeur, on utilise une pile. On empile le sommet de départ.

Parcours en profondeur : principe de l'algorithme

1. Dans le parcours en profondeur, on utilise une pile. On empile le sommet de départ.
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

1. Dans le parcours en profondeur, on utilise une pile. On empile le sommet de départ.
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,

Parcours en profondeur : principe de l'algorithme

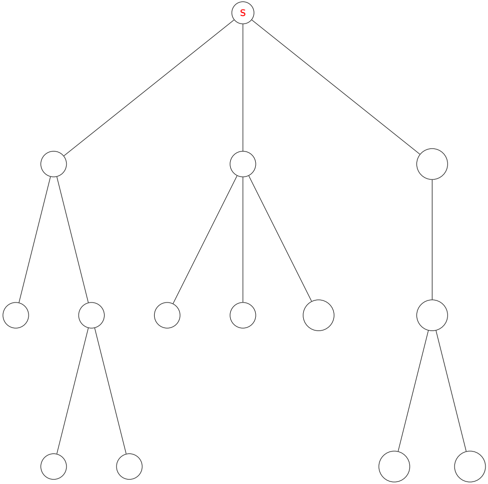
1. Dans le parcours en profondeur, on utilise une pile. On empile le sommet de départ.
2. Si le sommet de la pile présente des voisins qui ne sont pas dans la pile, ni déjà passés dans la pile :
   - alors on sélectionne l'un de ces voisins et on l'empile,
   - sinon on dépile (c'est à dire on supprime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

Parcours en profondeur : principe de l'algorithme

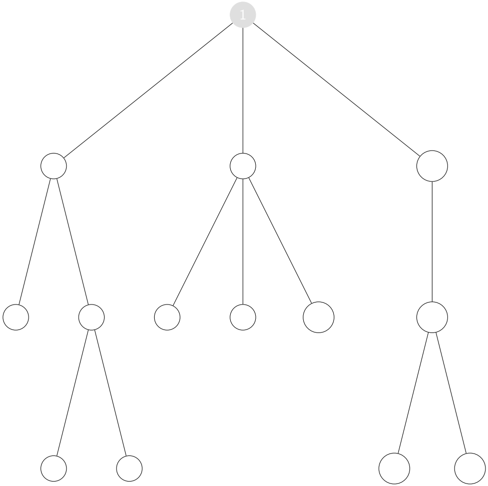
1. Dans le parcours en profondeur, on utilise une pile. On empile le sommet de départ.
2. Si le sommet de la pile présente des voisins qui ne sont pas dans la pile, ni déjà passés dans la pile :
   - alors on sélectionne l'un de ces voisins et on l'empile,
   - sinon on dépile (c'est à dire on supprime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

Parcours en profondeur d'un arbre

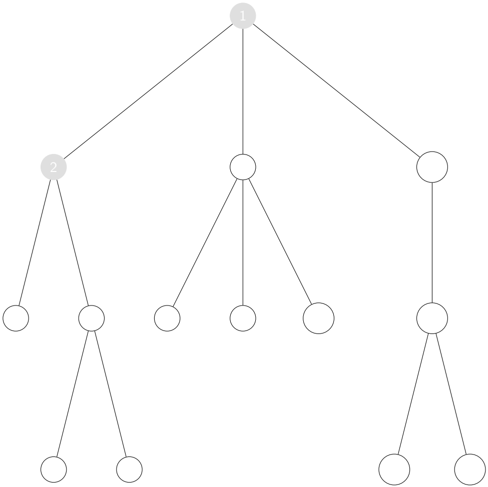
Parcourir en profondeur le graphe ci-dessous à partir du sommet s :



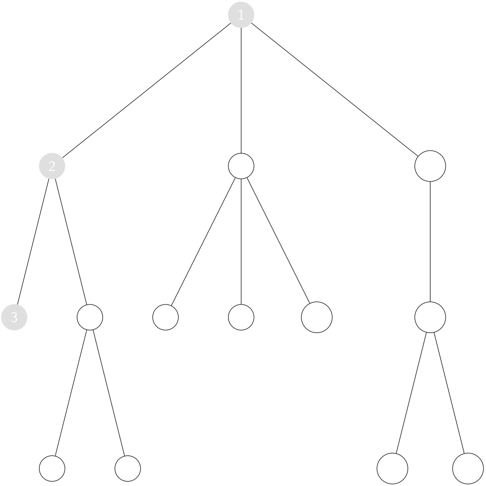
Parcours en profondeur d'un arbre



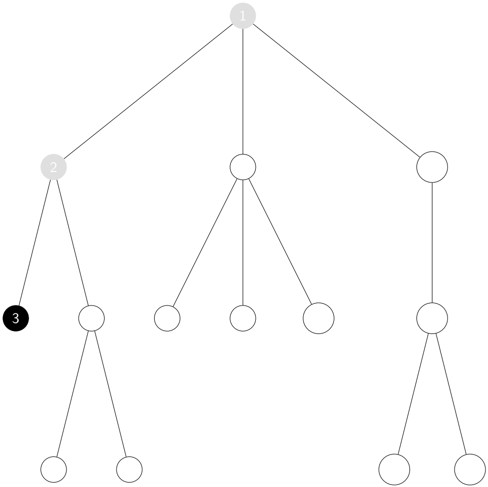
Parcours en profondeur d'un arbre



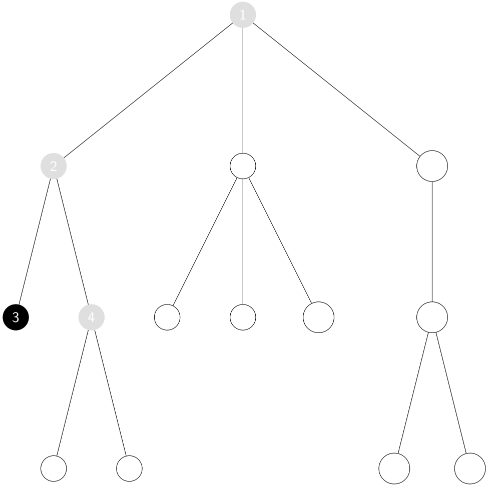
Parcours en profondeur d'un arbre



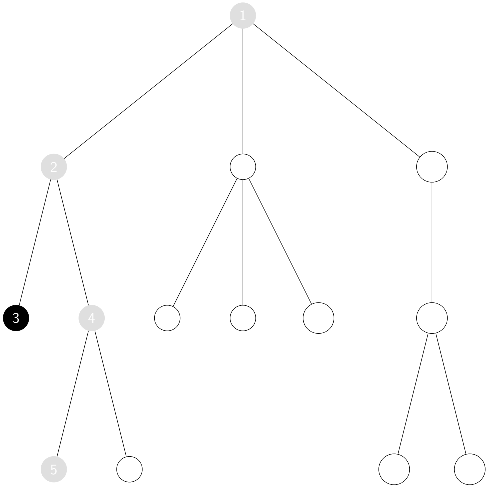
Parcours en profondeur d'un arbre



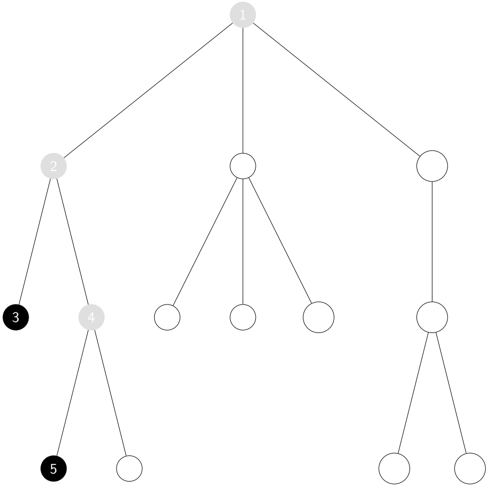
Parcours en profondeur d'un arbre



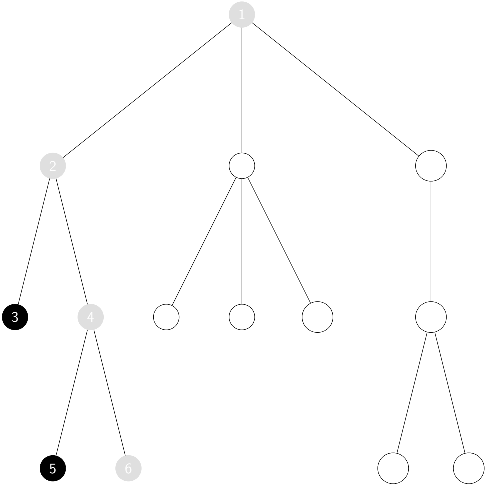
Parcours en profondeur d'un arbre



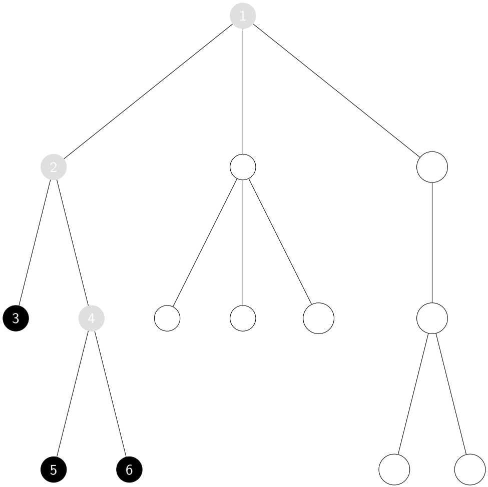
Parcours en profondeur d'un arbre



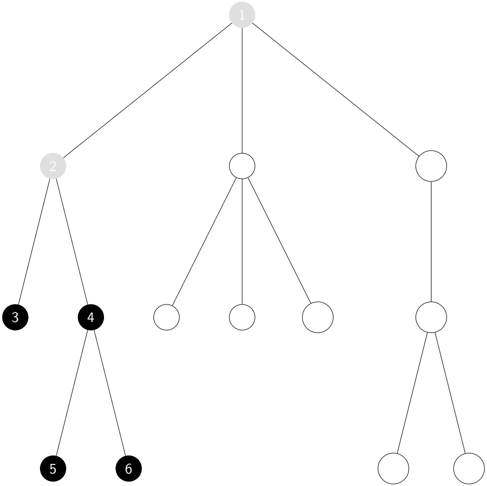
Parcours en profondeur d'un arbre



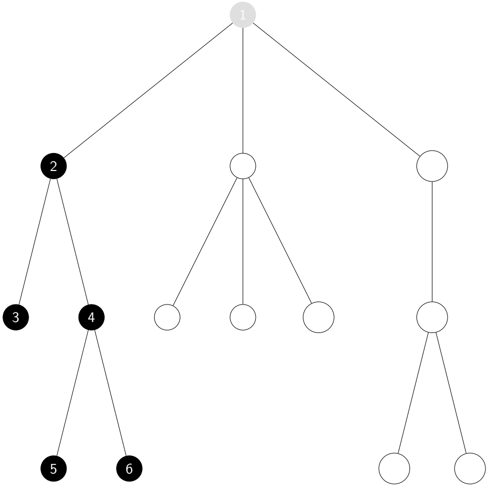
Parcours en profondeur d'un arbre



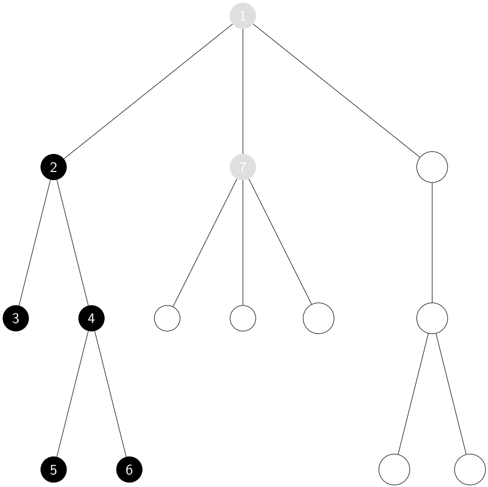
Parcours en profondeur d'un arbre



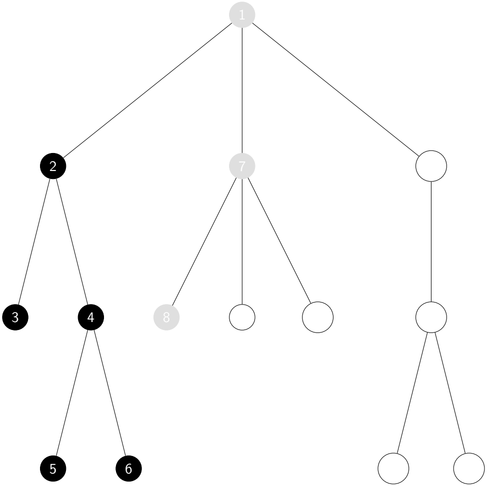
Parcours en profondeur d'un arbre



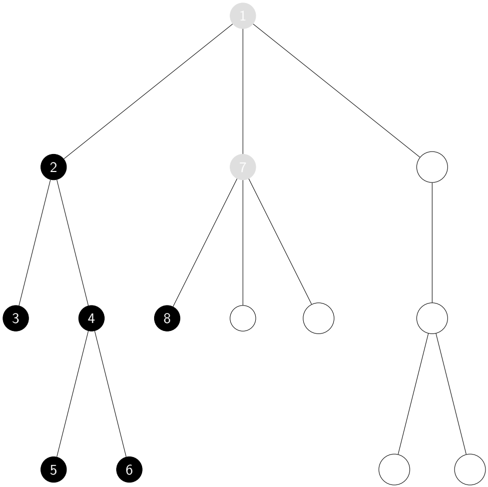
Parcours en profondeur d'un arbre



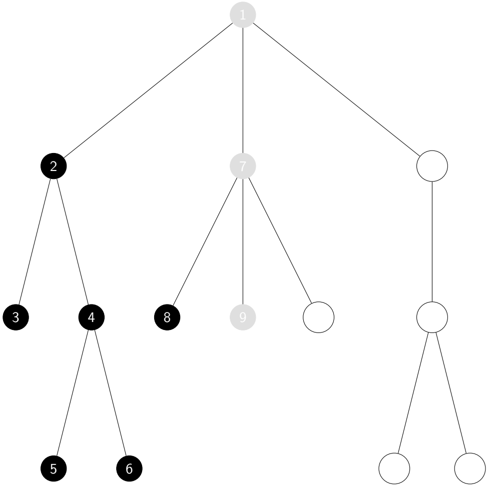
Parcours en profondeur d'un arbre



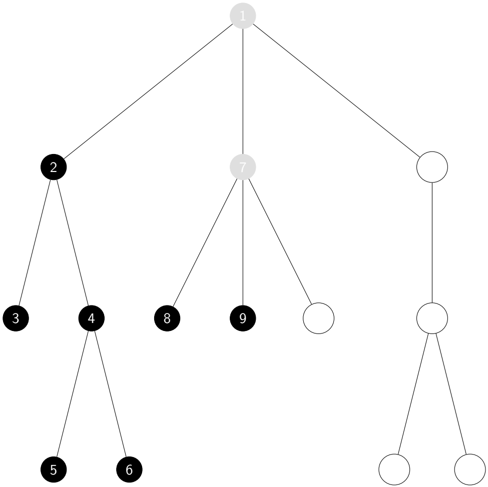
Parcours en profondeur d'un arbre



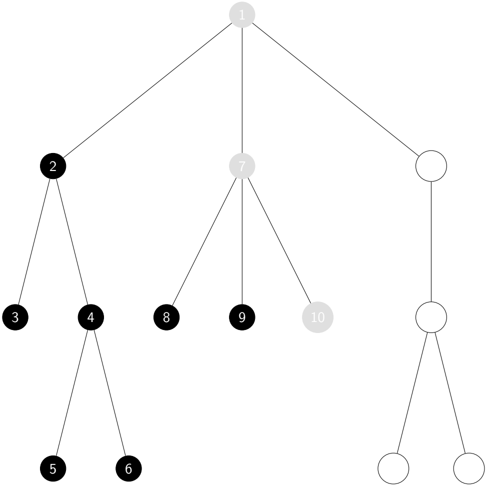
Parcours en profondeur d'un arbre



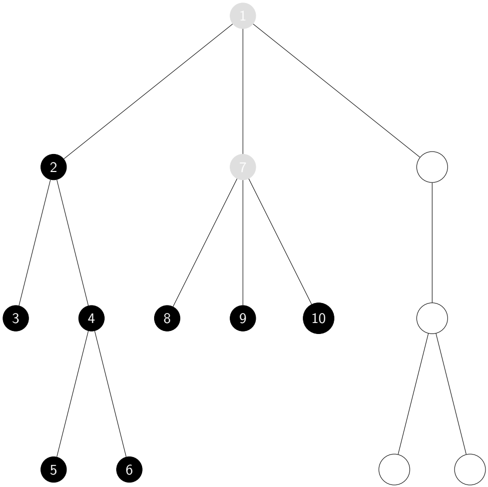
Parcours en profondeur d'un arbre



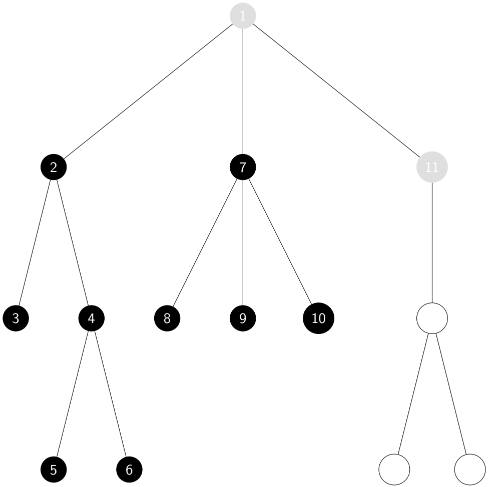
Parcours en profondeur d'un arbre



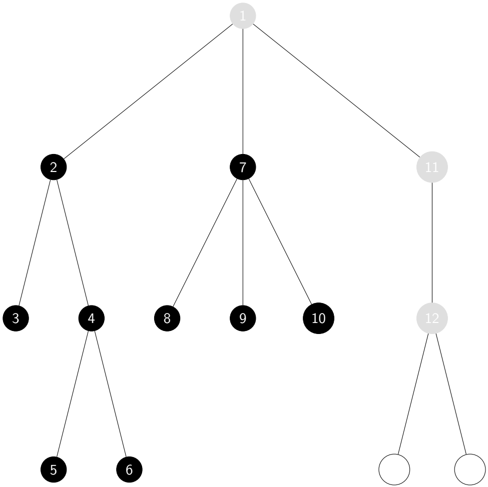
Parcours en profondeur d'un arbre



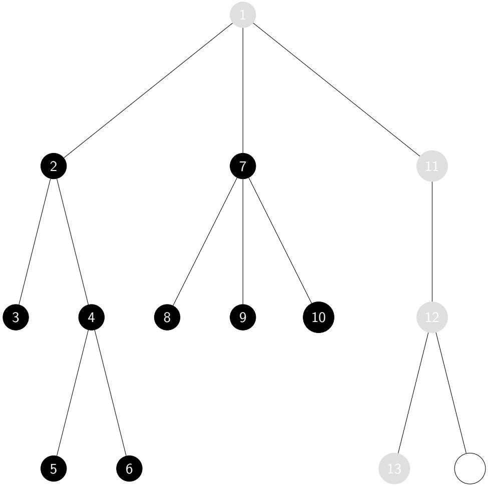
Parcours en profondeur d'un arbre



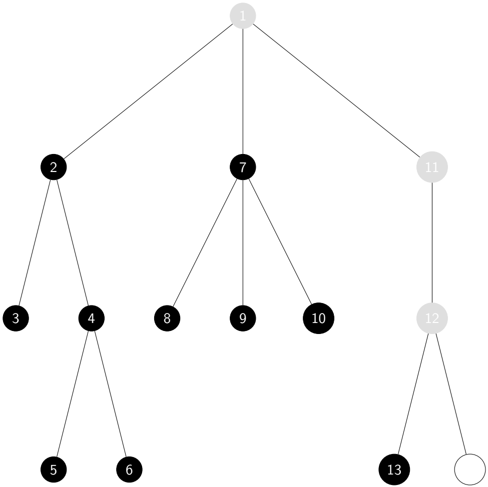
Parcours en profondeur d'un arbre



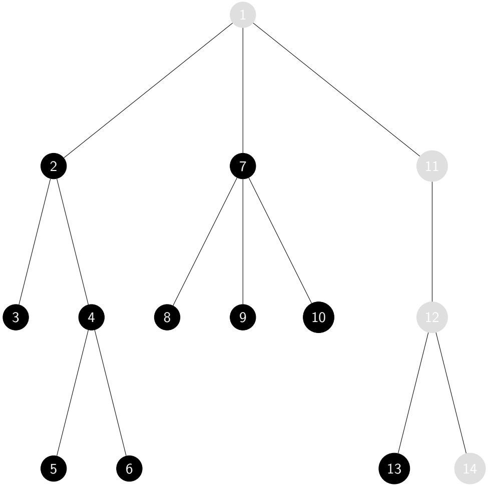
Parcours en profondeur d'un arbre



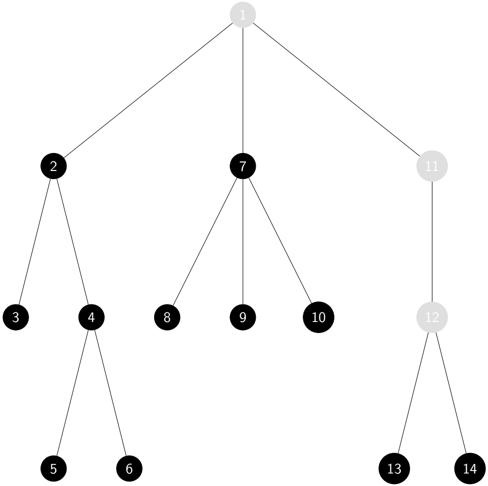
Parcours en profondeur d'un arbre



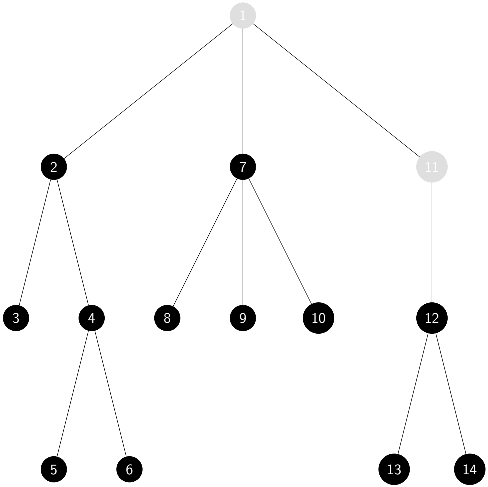
Parcours en profondeur d'un arbre



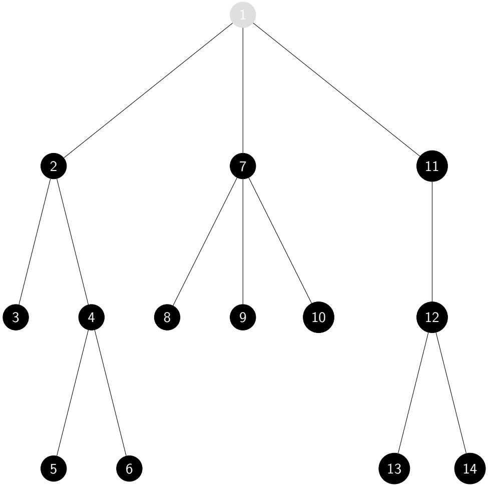
Parcours en profondeur d'un arbre



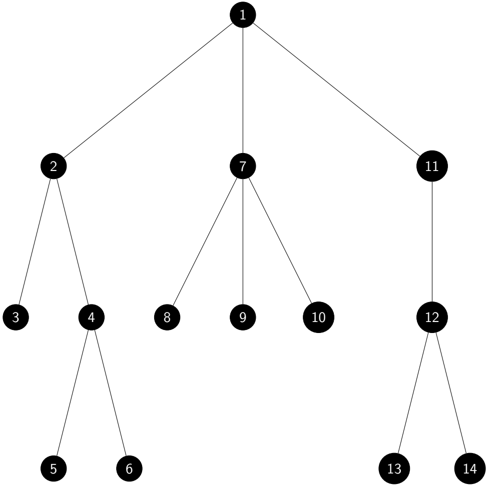
Parcours en profondeur d'un arbre



Parcours en profondeur d'un arbre

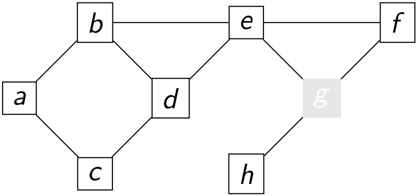


Parcours en profondeur d'un arbre



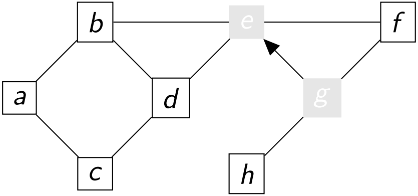
arbre DFS en python

Déroulement



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

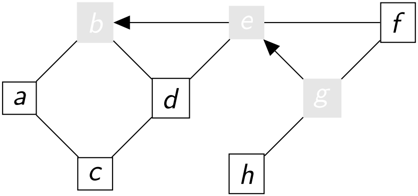
Déroulement



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

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

Déroulement



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

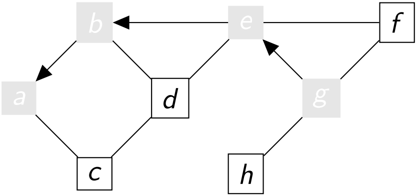
P={ g : None, e : g,b : e}

Q=[g,e,b]

Découverts=[g,e,b]

Fermés=[]

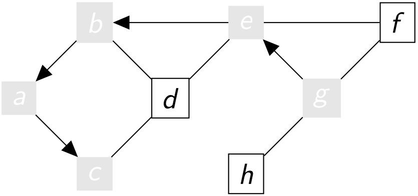
Déroulement



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,

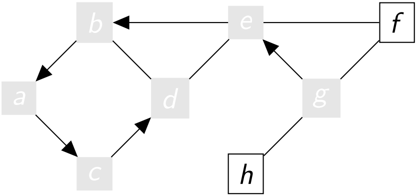
P={ g : None, e : g,b : e, a :b,c :a}

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

Découverts=[g,e,b,a,c]

Fermés=[]

Déroulement



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

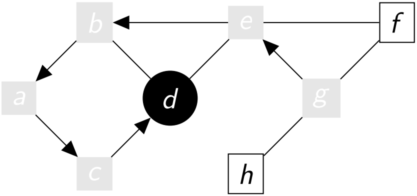
P={ g : None, e : g,b : e, a :b,c :a, d :c}

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

Découverts=[g,e,b,a,c,d]

Fermés=[]

Déroulement



u=d, R=[],

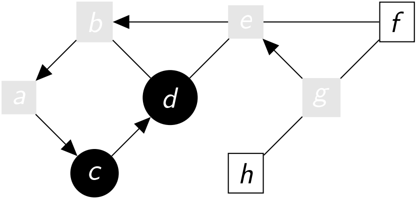
P={ g : None, e : g,b : e, a :b,c :a, d :c}

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

Découverts=[g,e,b,a,c,d]

Fermés=[d]

Déroulement



u=c, R=[],

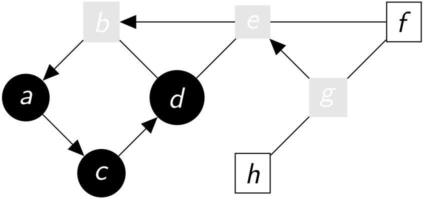
P={ g : None, e : g,b : e, a :b,c :a, d :c}

Q=[g,e,b,a]

Découverts=[g,e,b,a,c,d]

Fermés=[d,c]

Déroulement



u=a, R=[],

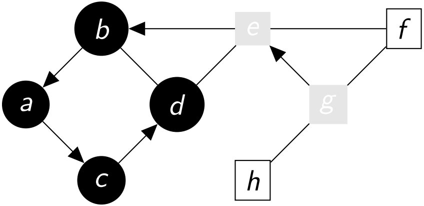
P={ g : None, e : g,b : e, a :b,c :a, d :c}

Q=[g,e,b]

Découverts=[g,e,b,a,c,d]

Fermés=[d,c,a]

Déroulement



u=b, R=[],

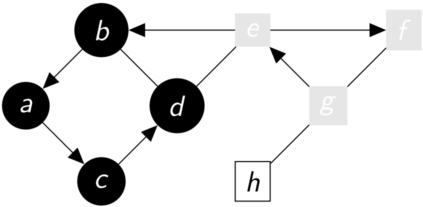
P={ g : None, e : g,b : e, a :b,c :a, d :c}

Q=[g,e]

Découverts=[g,e,b,a,c,d]

Fermés=[d,c,a,b]

Déroulement



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

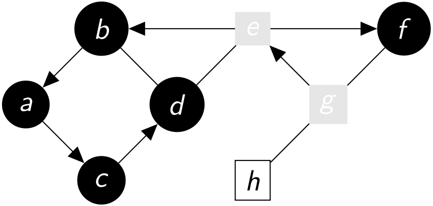
P={ g : None, e : g,b : e, a :b,c :a, d :c,f :e}

Q=[g,e,f]

Découverts=[g,e,b,a,c,d,f]

Fermés=[d,c,a,b]

Déroulement



u=f, R=[],

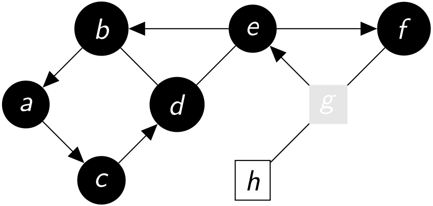
P={ g : None, e : g,b : e, a :b,c :a, d :c,f :e}

Q=[g,e]

Découverts=[g,e,b,a,c,d,f]

Fermés=[d,c,a,b,f]

Déroulement



u=e, R=[],

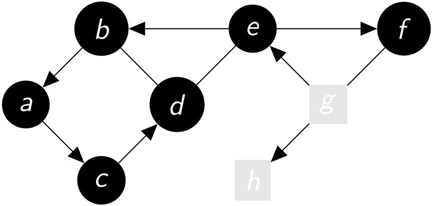
P={ g : None, e : g,b : e, a :b,c :a, d :c,f :e}

Q=[g]

Découverts=[g,e,b,a,c,d,f]

Fermés=[d,c,a,b,f,e]

Déroulement



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

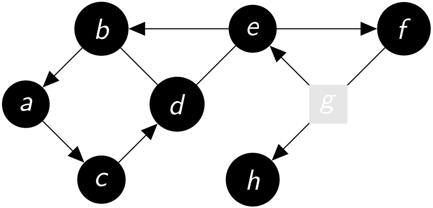
P={ g : None, e : g,b : e, a :b,c :a, d :c,f :e,h :g}

Q=[g,h]

Découverts=[g,e,b,a,c,d,f,h]

Fermés=[d,c,a,b,f,e]

Déroulement



u=h, R=[],

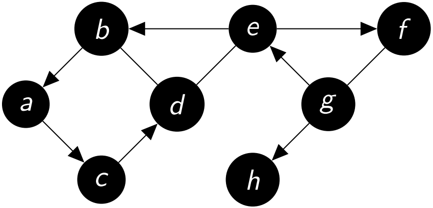
P={ g : None, e : g,b : e, a :b,c :a, d :c,f :e,h :g}

Q=[g]

Découverts=[g,e,b,a,c,d,f,h]

Fermés=[d,c,a,b,f,e,h]

Déroulement



u=g, R=[],

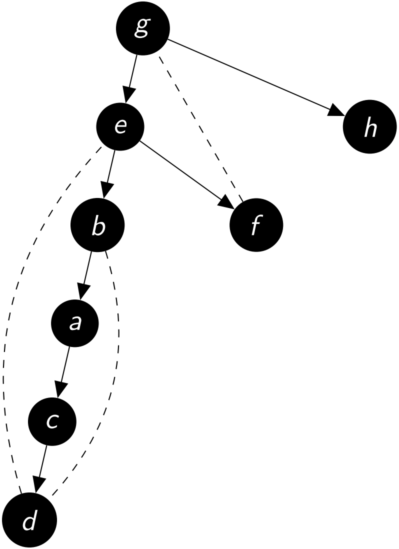
P={ g : None, e : g,b : e, a :b,c :a, d :c,f :e,h :g}

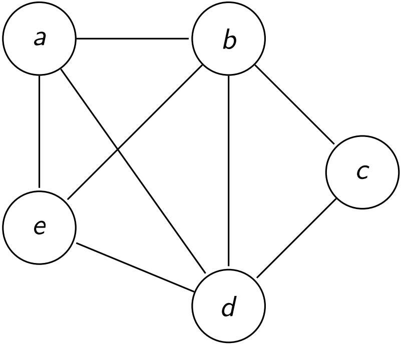
Q=[ ]

Découverts=[g,e,b,a,c,d,f,h]

Fermés=[d,c,a,b,f,e,h,g]

Arborescence associée au parcours




a
b
c
e
d

En Python,

```
def dfs(G,s) :
    P,Q={s :None},[s]
    while Q :
        u=Q[-1]
        R=[y for y in G[u] if y not in P]
        if R :
            v=random.choice(R)
            P[v]=u
            Q.append(v)
        else :
            Q.pop()
    return P
```

Plus court chemin

Étant donnés un graphe valué $G = (S, A, v)$ et un sommet a , on veut déterminer pour chaque sommet s la distance et un plus court chemin de a à s.

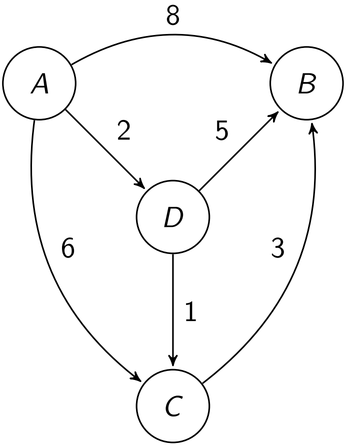
Plus court chemin

On note $d(x)$ la distance du plus court chemin entre a et x où x est un sommet de G.

1. $d(a) = 0$
2. Pour tout x admettant des antécédents y,

   d(x)=min(d(y)+ le poids de l'arête x,y) où y est un antécédent de x.

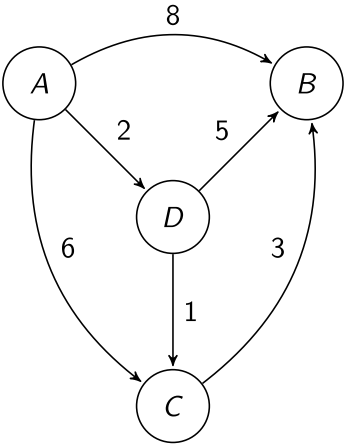
Plus court chemin



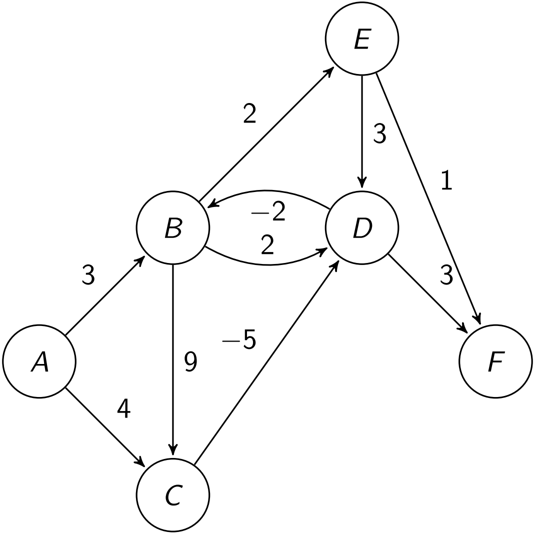
- Bellman- Ford -

On applique le principe précédent en explorant systématiquement tous les sommets et tous leurs successeurs. On s'arrête quand les valeurs des distances sont stabilisées.

Plus court chemin - Bellman- Ford - Exemple 1



Plus court chemin - Bellman- Ford - Exemple 2



- Bellman- Ford - A retenir :

Définition :

Un circuit **absorbant** est un circuit de valuation négative.

Remarque :

Si un graphe possède un circuit absorbant, alors il n'existe pas de plus courts chemins entre certains de ces sommets.

- Bellman- Ford - A retenir :

Propriété :

L'algorithme se termine (les valeurs se stabilisent) après au plus n passages dans la boucle principale, n étant le nombre de sommets du graphe.

- Bellman- Ford - A retenir :

Propriété :

L'algorithme se termine (les valeurs se stabilisent) après au plus n passages dans la boucle principale, n étant le nombre de sommets du graphe.

Corollaire :

L'algorithme permet de détecter la présence de circuits absorbants : si les valeurs des distances ne sont pas stabilisées après n passages de boucles, alors le graphe contient au moins un circuit absorbant.

Plus court chemin - Dijkstra -

L'algorithme de Dijkstra est un autre algorithme de recherche de distance et de plus court chemin.
Il est plus efficace que Bellman-Ford, mais ne fonctionne que dans le cas où toutes les valuations des arcs sont positives.

Plus court chemin - Dijkstra -

Principe :

Plus court chemin - Dijkstra -

Principe :

1. On construit petit à petit, à partir de a, un ensemble M de sommets marqués. Pour tout sommet marqué s, l'estimation $d(s)$ est égale à la distance $d(a, s)$.

Plus court chemin - Dijkstra -

Principe :

1. On construit petit à petit, à partir de a, un ensemble M de sommets marqués. Pour tout sommet marqué s, l'estimation $d(s)$ est égale à la distance $d(a, s)$.
2. A chaque étape, on sélectionne le (un) sommet non marqué x dont la distance estimé d(x) est la plus petite parmi tous les sommets non marqué.

Plus court chemin - Dijkstra -

Principe :

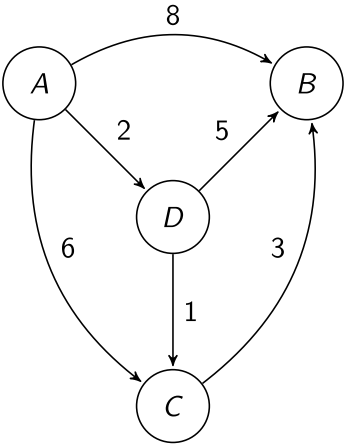
1. On construit petit à petit, à partir de a, un ensemble M de sommets marqués. Pour tout sommet marqué s, l'estimation $d(s)$ est égale à la distance $d(a, s)$.
2. A chaque étape, on sélectionne le (un) sommet non marqué x dont la distance estimé d(x) est la plus petite parmi tous les sommets non marqué.
3. On marque alors x (on rajoute x à M), puis on met à jour à partir de x les distances estimées des successeurs non marqués de x.

Plus court chemin - Dijkstra -

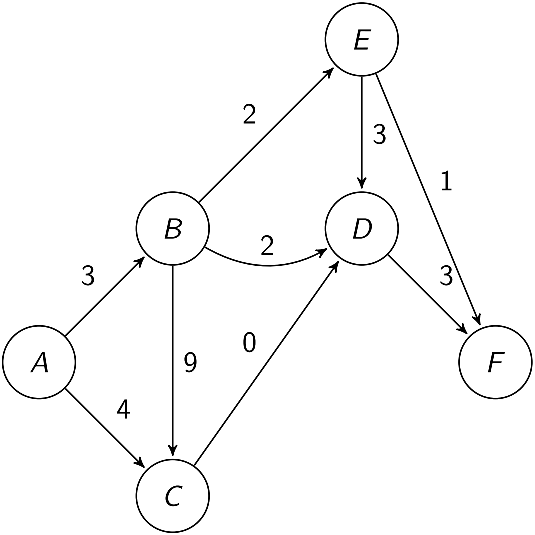
Principe :

1. On construit petit à petit, à partir de a, un ensemble M de sommets marqués. Pour tout sommet marqué s, l'estimation $d(s)$ est égale à la distance $d(a, s)$.
2. A chaque étape, on sélectionne le (un) sommet non marqué x dont la distance estimé d(x) est la plus petite parmi tous les sommets non marqué.
3. On marque alors x (on rajoute x à M), puis on met à jour à partir de x les distances estimées des successeurs non marqués de x.
4. On recommence, jusqu'à épuisement des sommets non marqués.

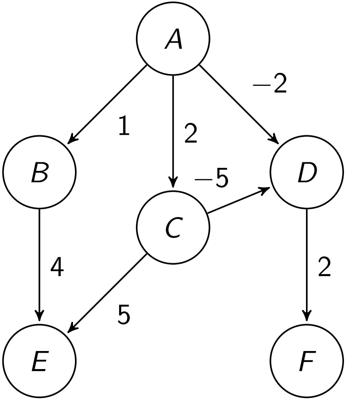
Plus court chemin - Dijkstra - Exemple 1



Plus court chemin - Dijkstra - Exemple 2



Plus court chemin - Dijkstra - Exemple 3



L'algorithme de Dijkstra ne fonctionne que pour des poids positifs car une fois qu'un sommet est marqué on ne peut changer ce marquage dans la suite de l'algorithme

BILAN (Chemin le plus court)

BILAN (Chemin le plus court)

- **Dijkstra :** Plus court chemin d'un sommet à tous les autres, uniquement poids positif, complexité $\mathcal{O}(m + n.\log(n))$

BILAN (Chemin le plus court)

- **Dijkstra :** Plus court chemin d'un sommet à tous les autres, uniquement poids positif, complexité $\mathcal{O}(m + n.\log(n))$
- **Bellman :** Plus court chemin d'un sommet à tous les autres, poids négatifs autorisés, complexité $\mathcal{O}(mn)$

BILAN (Chemin le plus court)

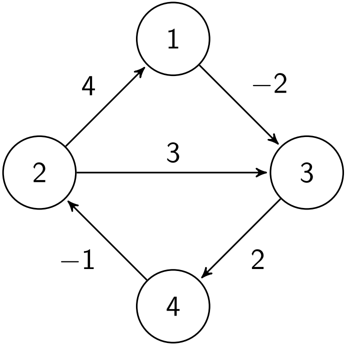
- **Dijkstra :** Plus court chemin d'un sommet à tous les autres, uniquement poids positif, complexité $\mathcal{O}(m + n.\log(n))$
- **Bellman :** Plus court chemin d'un sommet à tous les autres, poids négatifs autorisés, complexité $\mathcal{O}(mn)$
- **Floyd-Warshall :** Plus court chemin de **TOUS** les sommets à **TOUS** les autres, poids négatifs autorisés, complexité $\mathcal{O}(n^3)$

Plus court chemin - Floyd-Warshall -

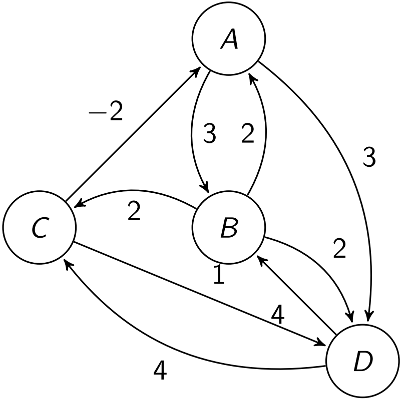
Entrées: G : *graphe*

Sorties: Matrice de plus courts chemins

pour $k = 1$ à n **faire**
  pour $i = 1$ à n et $i \neq k$ **faire**
    si $l_{ik} + l_{ki} < 0$ **alors**
      FIN – circuit négatif
    finsi
    si $l_{ik} \neq \infty$ **alors**
      pour $j = 1$ à n et $j \neq i$ **faire**
        $\gamma = l_{ik}$
        si $\gamma + l_{kj} < l_{ij}$ **alors**
          $l_{ij} \leftarrow \gamma + l_{kj}$
          $p_{ij} \leftarrow p_{kj}$
        finsi
      fin pour
    finsi
  fin pour
fin pour



Plus court chemin - Floyd-Warshall -



Arbres

- Définition :
  On appelle source de G les sommets n'ayant aucun . . . . . . et puits les sommets n'ayant aucun . . . .

Arbres

- Définition :
  On appelle source de G les sommets n'ayant aucun . . . . . . et puits les sommets n'ayant aucun . . . .

- **Définition :**
  On dit qu'un graphe non orienté est acyclique si il ne contient pas de cycle.

Arbres

- Définition :
  On appelle source de G les sommets n'ayant aucun . . . . . . et puits les sommets n'ayant aucun . . . .

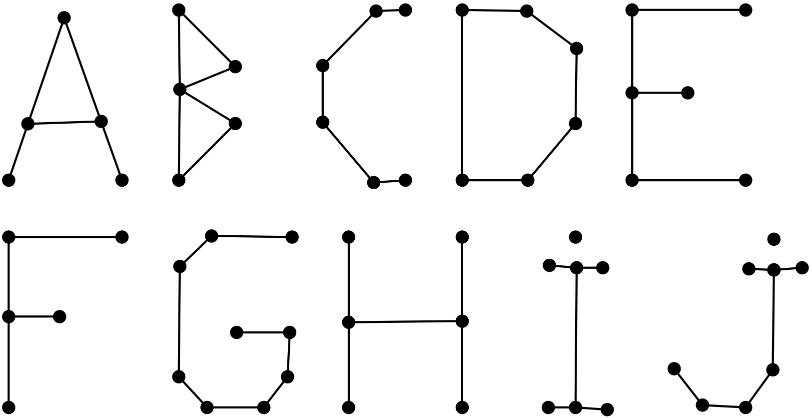
- **Définition :**
  On dit qu'un graphe non orienté est acyclique si il ne contient pas de cycle.
- **Définition :**
  Un **arbre** est un graphe connexe sans cycle.

Arbres

- Définition :
  On appelle source de G les sommets n'ayant aucun . . . . . . et puits les sommets n'ayant aucun . . . .

- **Définition :**
  On dit qu'un graphe non orienté est acyclique si il ne contient pas de cycle.
- **Définition :**
  Un **arbre** est un graphe connexe sans cycle.
- **Définition :**
  Une **forêt** est un graphe sans cycle. Elle est composée d'arbres.

Parmi les graphes suivants, lesquels sont des arbres ?



Problème (ARPM) :

Pour un graphe donné, le problème du graphe recouvrant de poids minimum, est un problème dans lequel nous cherchons à trouver un arbre dont les sommets sont les sommets du graphe et la somme des poids des arêtes est minimale.

Exemple :

Soit G un graphe ayant n sommets.
Les propositions suivantes sont équivalents :

- G est un arbre,
- G est acyclique et a $n - 1$ arêtes,
- G est connexe et a $n - 1$ arêtes,
- G est acyclique et en ajoutant une arête entre deux sommets non adjacents, on crée un cycle et un seule,
- G est connexe et, en supprimant une arête quelconque, il n'est plus connexe,
- tout couple de sommets est relié par une chaîne élémentaire et une seule.

- **Définition :**
  Un **arbre couvrant** d'un graphe G non orienté est un graphe T tel que :

- **Définition :**
  Un **arbre couvrant** d'un graphe G non orienté est un graphe T tel que :
  - Les sommets de T sont égaux aux sommets de G.

- **Définition :**
  Un **arbre couvrant** d'un graphe G non orienté est un graphe T tel que :
  - Les sommets de T sont égaux aux sommets de G.
  - T est un arbre.

- **Définition :**
  Un **arbre couvrant** d'un graphe G non orienté est un graphe T tel que :
  - Les sommets de T sont égaux aux sommets de G.
  - T est un arbre.
- Propriété :
  Tout graphe connexe admet un arbre couvrant.