

# ***Recueil d'exercices d'Algorithmes et Structures de Données (ASD)***

## ***Partie 3 : structures non linéaires***

Version 0.6  
16 juin 2023

© Olivier Cuisenaire, 2023

## Table des matières

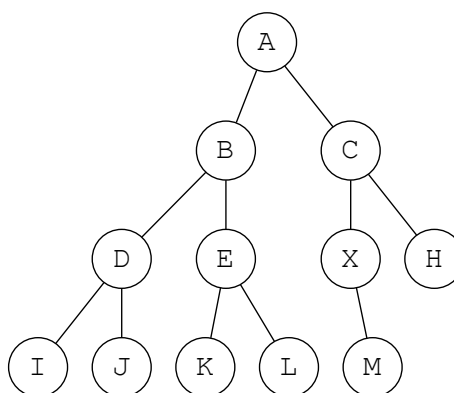
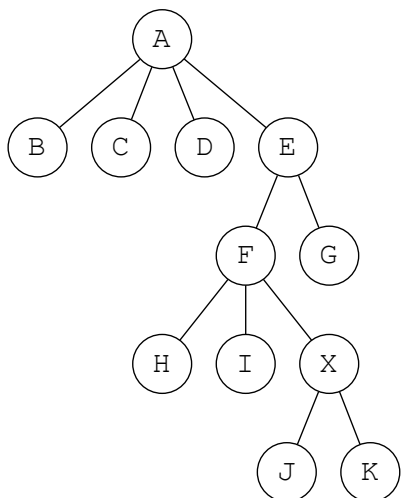
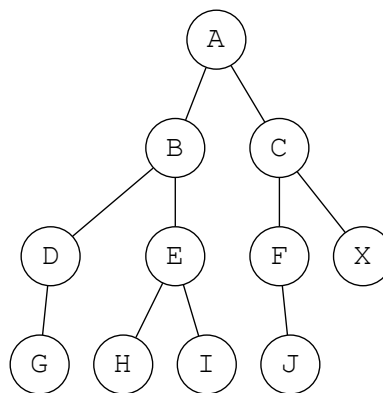
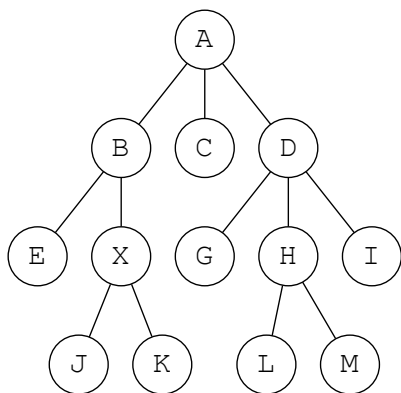
Chapitre 5 : Arbres .....	1
Exercice 5.1 Nomenclature .....	1
Exercice 5.2 Parcours.....	2
Exercice 5.3 Listes imbriquées (1).....	3
Exercice 5.4 Listes imbriquées (2).....	3
Exercice 5.5 Listes imbriquées (3).....	4
Exercice 5.6 Nomenclature (2) .....	4
Exercice 5.7 Parcours arbre binaire (1).....	5
Exercice 5.8 Parcours arbre binaire (2).....	5
Exercice 5.9 Insertions dans un arbre binaire de recherche.....	6
Exercice 5.10 Suppressions d'un arbre binaire de recherche .....	6
Exercice 5.11 Parcours d'un arbre binaire de recherche.....	6
Exercice 5.12 Equilibre.....	7
Exercice 5.13 Insertion dans un arbre AVL.....	8
Exercice 5.14 Suppression dans un arbre AVL .....	8
Solutions .....	10
Chapitre 6 : Graphes .....	26
Exercice 6.1 Parcours.....	26
Exercice 6.2 Dijkstra.....	28
Exercice 6.3 Tri topologique.....	30
Exercice 6.4 Composantes fortement connexes.....	31
Solutions .....	36
Chapitre 7 : Tables de hachage .....	44
Exercice 7.1 Résolution de collisions par chaînage.....	44
Exercice 7.2 Résolution de collisions par chaînage (2) .....	44
Exercice 7.3 Résolution de collisions par sondage linéaire.....	45
Solutions .....	46

## Chapitre 5 : Arbres

### Exercice 5.1 Nomenclature

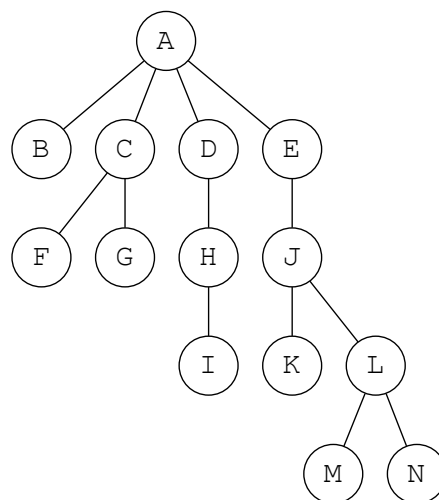
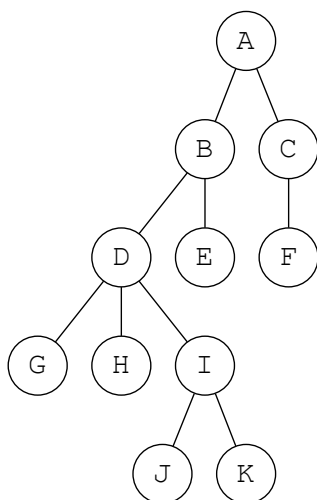
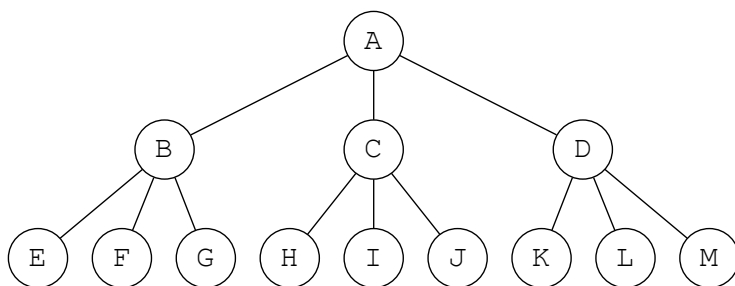
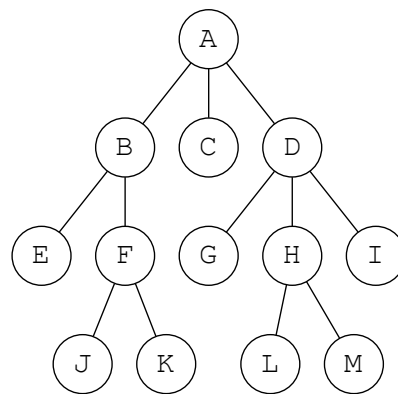
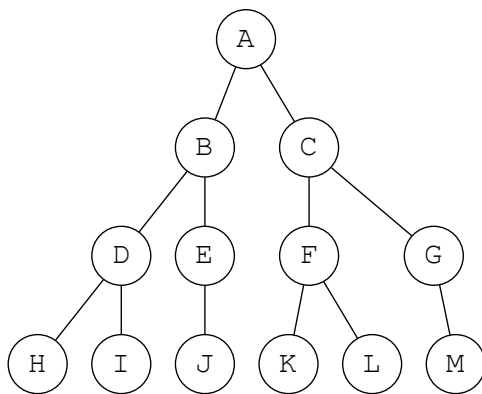
Pour chacun des arbres ci-dessous, indiquez

1. Son degré
2. Sa racine
3. Sa hauteur
4. La liste de ses nœuds internes
5. La liste de ses feuilles
6. Le chemin du nœud X et son niveau
7. S'il est vide, plein, complet, dégénéré ou rien de cela



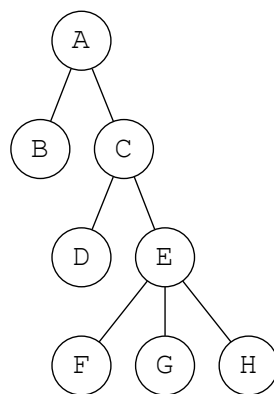
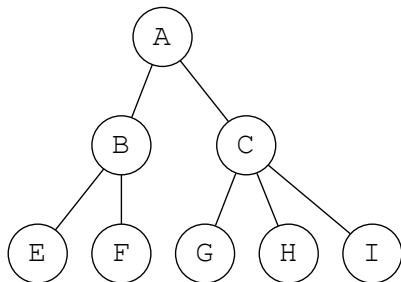
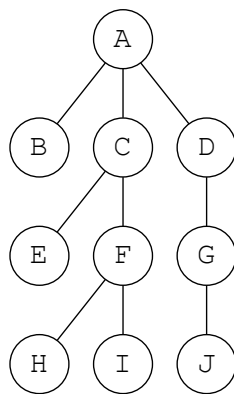
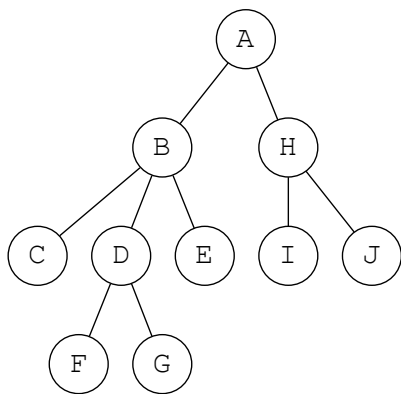
## Exercice 5.2 Parcours

Effectuez les parcours en profondeur, en profondeur post-ordonné et en largeur des arbres suivants



### Exercice 5.3 Listes imbriquées (1)

Donnez la représentation sous forme de listes imbriquées des arbres suivants



### Exercice 5.4 Listes imbriquées (2)

Dessinez les arbres dont les listes imbriquées sont

1.  $(A(B(D), C))$
2.  $(A(B, C(E, F), D(G)))$
3.  $(A(B(E, F), C(G, H, I)))$
4.  $(A(B(C), D, E(F, G)))$

## Exercice 5.5 Listes imbriquées (3)

Soit un arbre donc les nœuds utilisent la structure ci-dessous.

```
struct Noeud {  
    string etiquette;  
    Noeud* aine;  
    Noeud* puine;  
};
```

Ecrivez la fonction `operator<<` permettant d'afficher sous forme de listes imbriquées l'arbre dont on passe en paramètre le pointeur sur son Noeud racine. Utilisez comme base le code disponible ici : <https://gist.github.com/ocuisenaire/a8ddbc46e141e5ff86e12b0f34a87869> . Le programme résultat doit afficher

```
A(B(C,D(F,G),E),H(I,J))  
A(B,C(E,F(H,I)),D(G(J)))  
A(B(E,F),C(G,H,I))  
A(B,C(D,E(F,G,H)))
```

## Exercice 5.6 Nomenclature (2)

Soit un arbre donc les nœuds utilisent la même structure qu'à l'exercice précédent, écrivez les fonctions `taille`, `hauteur`, `degre`, `feuilles` et `internes` de sorte que le programme

```
Noeud* n;  
cin >> n;  
cout << "  taille :           " << taille(n) << endl;  
cout << "  hauteur :          " << hauteur(n) << endl;  
cout << "  degre :               " << degre(n) << endl;  
cout << "  feuilles :             " << feuilles(n) << endl;  
cout << "  noeuds internes :      " << internes(n) << endl;
```

Affiche le résultat suivant lorsque l'utilisateur entre `A(B,C(E,F(H,I)),D(G(J)))`

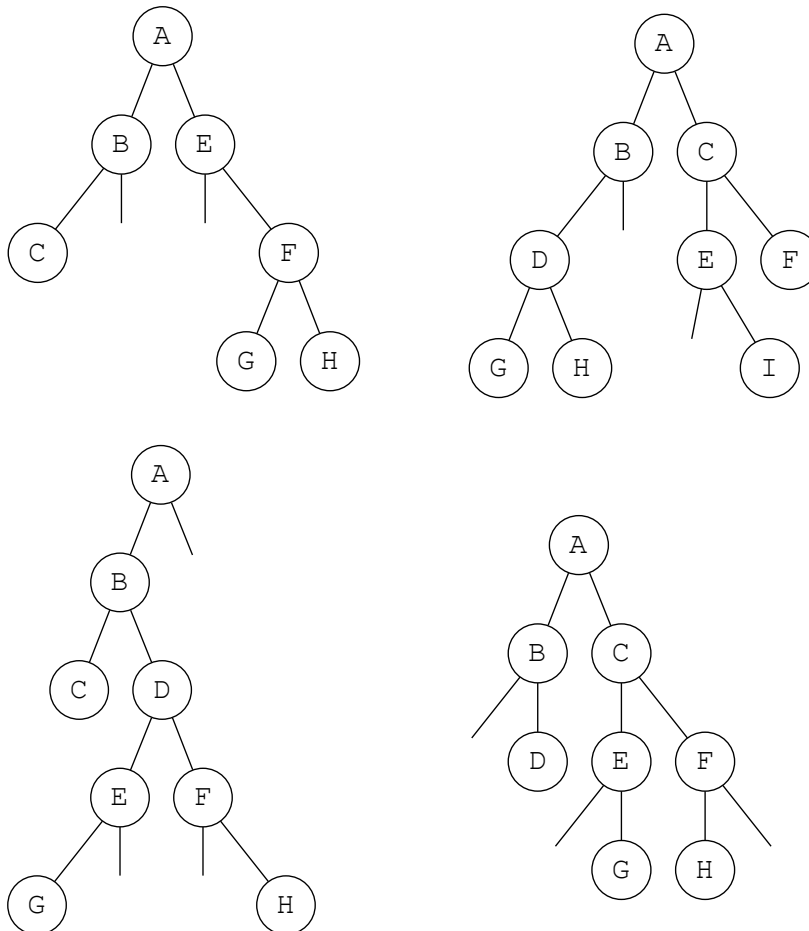
```
taille :           10  
hauteur :          4  
degre :            3  
feuilles :         B E H I J  
noeuds internes : A C F D G
```

Utilisez comme base le code disponible ici :

<https://gist.github.com/ocuisenaire/8d213365f6f9e5b4a1a231df309778aa>

### Exercice 5.7 Parcours arbre binaire (1)

Effectuer les parcours en pré-ordre, symétrique, et en post-ordre des arbres binaires suivants



### Exercice 5.8 Parcours arbre binaire (2)

Dessinez les arbres binaires dont vous connaissez deux parcours

1. pre : A B D H I E K C F G  
sym : H D I B K E A C G F
2. sym : F G A E C D B H  
post : G F A D C H B E
3. pre : G A B D I C F H E  
sym : B A D I G H F C E
4. sym : A G D B E F H C  
post : A D E B G C H F

### Exercice 5.9 Insertions dans un arbre binaire de recherche

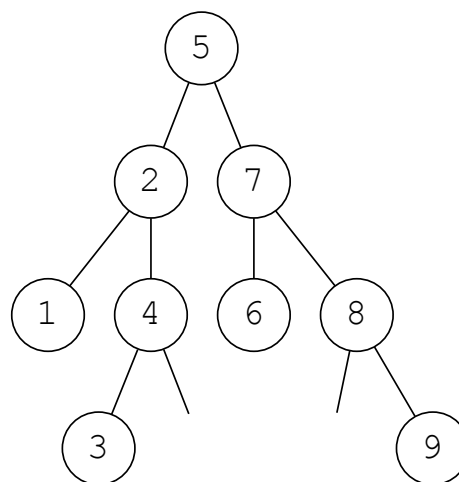
Dessinez les arbres binaires de recherche (non équilibrés) obtenus en insérant dans un arbre vide les clés suivantes dans cet ordre

- 5, 3, 1, 6, 7, 4, 8, 2, 9
- 4, 7, 2, 1, 5, 9, 6, 8, 3
- 1, 2, 3, 8, 5, 6, 4, 7, 9
- 5, 3, 9, 7, 6, 8, 2, 4, 1

### Exercice 5.10 Suppressions d'un arbre binaire de recherche

Dessinez les arbres binaires de recherche (non équilibrés) obtenus en supprimant l'élément mentionné de l'arbre ci-dessous. Pour chaque élément, partez de l'arbre original, pas du résultat précédent. En cas de suppression de Hibbard, indiquez les deux solutions possibles selon le côté choisi.

- Supprimer la clé 4
- Supprimer la clé 8
- Supprimer la clé 7
- Supprimer la clé 5



### Exercice 5.11 Parcours d'un arbre binaire de recherche

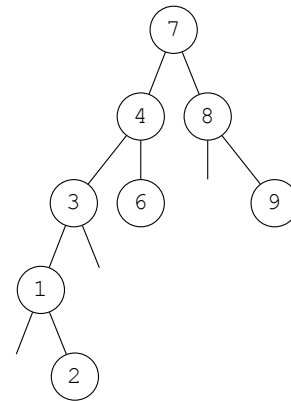
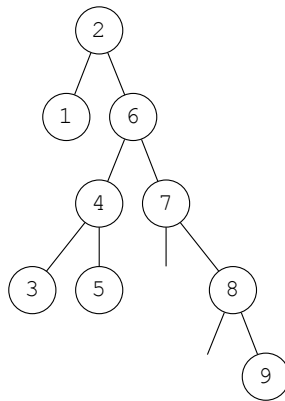
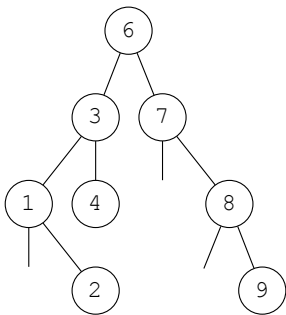
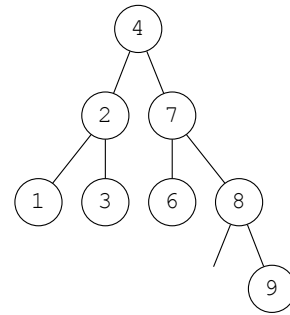
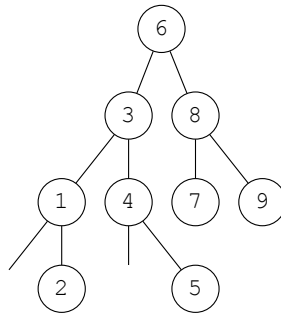
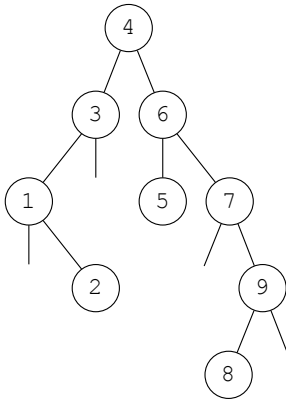
Dessinez les arbres binaires de recherche dont vous connaissez un parcours

- Pré-ordre : 4 3 1 2 6 5 7 9 8
- Post-ordre : 3 2 4 6 8 9 7 5 1
- Pré-ordre : 6 3 1 2 4 7 8 9
- Post-ordre : 2 1 3 6 4 9 8 7
- Pré-ordre : 4 2 1 3 5 6
- Post-ordre : 1 2 4 6 8 7 5 3



## Exercice 5.12 Equilibre

Les arbres suivants sont-ils équilibrés ? Si ce n'est pas le cas, indiquez quels nœuds sont déséquilibrés (sous-arbres de hauteurs différent de plus de 1).



### Exercice 5.13 Insertion dans un arbre AVL

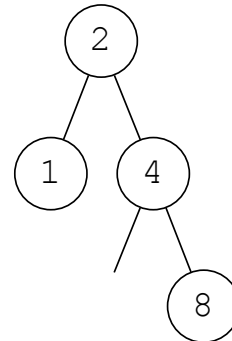
1. En partant à chaque fois de l'arbre AVL ci-contre

- Insérez la clé 9
- Insérez la clé 7

a)

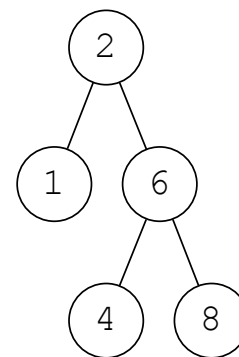


b)



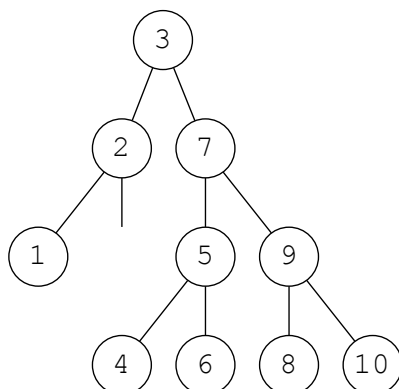
2. En partant à chaque fois de l'arbre AVL ci-contre

- Insérez la clé 3
- Insérez la clé 5
- Insérez la clé 7
- Insérez la clé 9

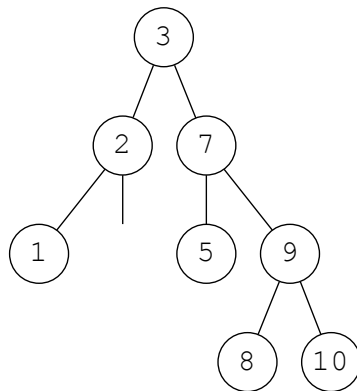


### Exercice 5.14 Suppression dans un arbre AVL

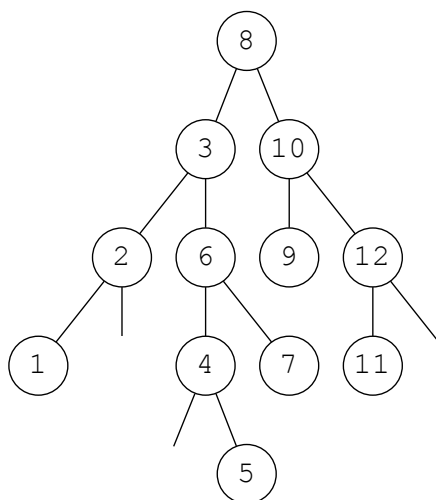
1. Supprimez la clé 2 de l'arbre suivant



2. Donnez les 2 arbres pouvant résulter de la suppression de la clé 3 de l'arbre suivant

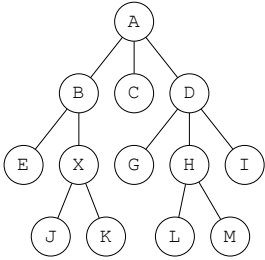
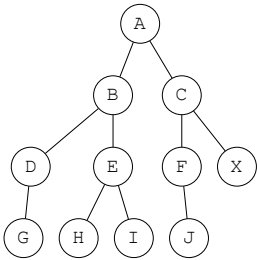
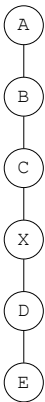
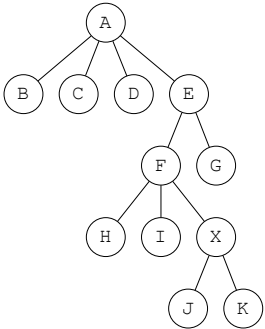


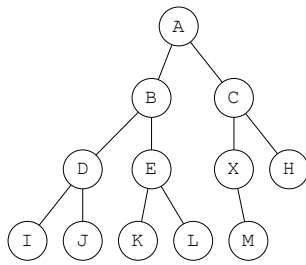
3. Supprimez la clé 9 de l'arbre suivant



## Solutions

### Exercice 5.1 Nomenclature

	<p>Degré 3, racine A, hauteur 4</p> <p>Nœuds internes : A,B,D,X,H</p> <p>Feuilles : C,E,G,I,J,K,L,M</p> <p>Nœud X de chemin (A,B,X) et de niveau 3</p> <p>L'arbre n'est ni vide, ni plein, ni complet ni dégénéré</p>
	<p>Degré 2, racine A, hauteur 4</p> <p>Nœuds internes : A,B,C,D,E,F</p> <p>Feuilles : X,G,H,I,J</p> <p>Nœud X de chemin (A,C,X) et de niveau 3</p> <p>L'arbre est plein, mais pas complet (le nœud D n'a qu'un enfant)</p>
	<p>Degré 1, racine A, hauteur 6</p> <p>Nœuds internes : A,B,C,X,D</p> <p>Feuilles : E</p> <p>Nœud X de chemin (A,B,C,X) et de niveau 4</p> <p>L'arbre est plein, complet et dégénéré</p>
	<p>Degré 4, racine A, hauteur 5</p> <p>Nœuds internes : A,E,F,X</p> <p>Feuilles : B,C,D,G,H,I,J,K</p> <p>Nœud X de chemin (A,E,F,X) et de niveau 4</p> <p>L'arbre n'est ni vide, ni plein, ni complet ni dégénéré</p>



Degré 2, racine A, hauteur 4

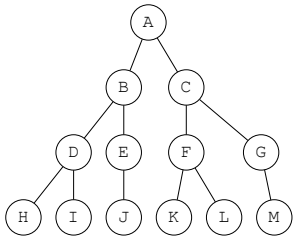
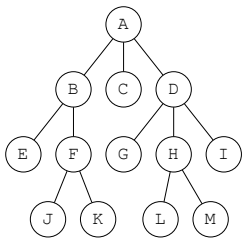
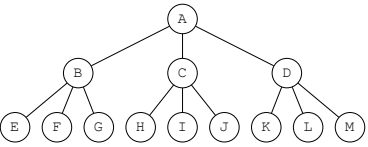
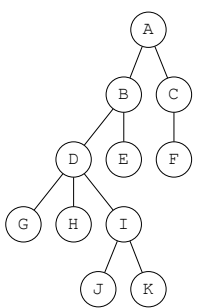
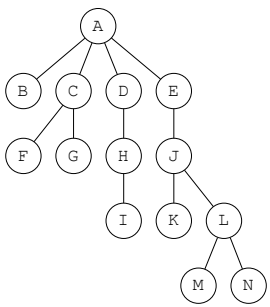
Nœuds internes : A,B,C,D,E,X

Feuilles : H,I,J,K,L,M

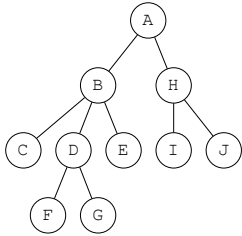
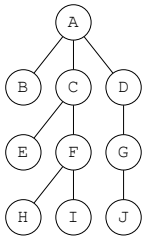
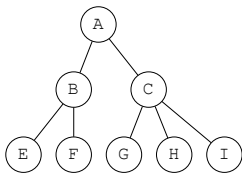
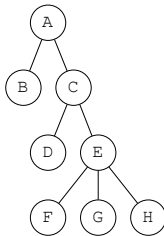
Nœud X de chemin (A,C,X) et de niveau 3

L'arbre est complet (et donc également plein), mais pas dégénéré

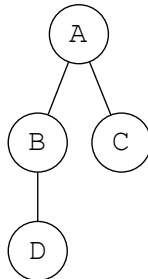
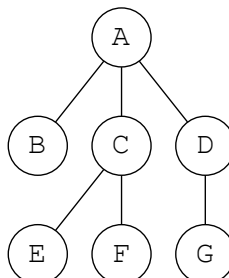
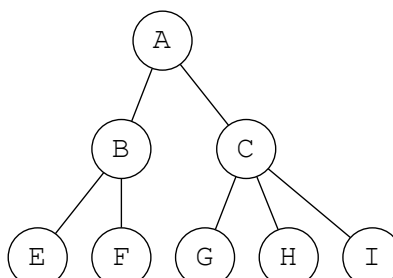
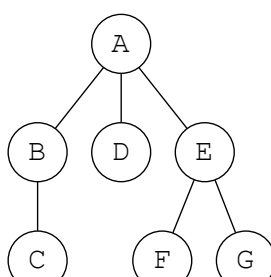
## Exercice 5.2 Parcours

 <pre> graph TD     A((A)) --- B((B))     A --- C((C))     B --- D((D))     B --- E((E))     C --- F((F))     C --- G((G))     D --- H((H))     D --- I((I))     E --- J((J))     F --- K((K))     F --- L((L))     G --- M((M)) </pre>	<p>A B D H I E J C F K L G M : profondeur H I D J E B K L F M G C A : post-ordonne A B C D E F G H I J K L M : largeur</p>
 <pre> graph TD     A((A)) --- B((B))     A --- C((C))     A --- D((D))     B --- E((E))     B --- F((F))     C --- G((G))     D --- H((H))     D --- I((I))     F --- J((J))     F --- K((K))     H --- L((L))     H --- M((M)) </pre>	<p>A B E F J K C D G H L M I : profondeur E J K F B C G L M H I D A : post-ordonne A B C D E F G H I J K L M : largeur</p>
 <pre> graph TD     A((A)) --- B((B))     A --- C((C))     A --- D((D))     B --- E((E))     B --- F((F))     B --- G((G))     C --- H((H))     C --- I((I))     D --- J((J))     D --- K((K))     D --- L((L))     F --- M((M)) </pre>	<p>A B E F G C H I J D K L M : profondeur E F G B H I J C K L M D A : post-ordonne A B C D E F G H I J K L M : largeur</p>
 <pre> graph TD     A((A)) --- B((B))     A --- C((C))     B --- D((D))     B --- E((E))     C --- F((F))     D --- G((G))     D --- H((H))     D --- I((I))     I --- J((J))     I --- K((K)) </pre>	<p>A B D G H I J K E C F : profondeur G H J K I D E B F C A : post-ordonne A B C D E F G H I J K : largeur</p>
 <pre> graph TD     A((A)) --- B((B))     A --- C((C))     A --- D((D))     A --- E((E))     B --- F((F))     B --- G((G))     C --- H((H))     D --- I((I))     E --- J((J))     E --- K((K))     J --- L((L))     L --- M((M))     L --- N((N)) </pre>	<p>A B C F G D H I E J K L M N : profondeur B F G C I H D K M N L J E A : post-ordonne A B C D E F G H J I K L M N : largeur</p>

### Exercice 5.3 Listes imbriquées (1)

 <pre> graph TD     A((A)) --- B((B))     A --- H((H))     B --- C((C))     B --- D((D))     B --- E((E))     D --- F((F))     D --- G((G))     H --- I((I))     H --- J((J))         </pre>	(A(B(C,D(F,G),E),H(I,J)))
 <pre> graph TD     A((A)) --- B((B))     A --- C((C))     A --- D((D))     C --- E((E))     C --- F((F))     F --- H((H))     F --- I((I))     D --- G((G))     D --- J((J))         </pre>	(A(B,C(E,F(H,I)),D(G(J))))
 <pre> graph TD     A((A)) --- B((B))     A --- C((C))     B --- E((E))     B --- F((F))     C --- G((G))     C --- H((H))     C --- I((I))         </pre>	(A(B(E,F),C(G,H,I)))
 <pre> graph TD     A((A)) --- B((B))     A --- C((C))     C --- D((D))     C --- E((E))     E --- F((F))     E --- G((G))     E --- H((H))         </pre>	(A(B,C(D,E(F,G,H))))

## Exercice 5.4 Listes imbriquées (2)

(A(B(D),C))	 <pre> graph TD     A((A)) --- B((B))     A --- C((C))     B --- D((D))         </pre>
(A(B,C(E,F),D(G)))	 <pre> graph TD     A((A)) --- B((B))     A --- C((C))     A --- D((D))     C --- E((E))     C --- F((F))     D --- G((G))         </pre>
(A(B(E,F),C(G,H,I)))	 <pre> graph TD     A((A)) --- B((B))     A --- C((C))     B --- E1((E))     B --- F1((F))     C --- G1((G))     C --- H1((H))     C --- I1((I))         </pre>
(A(B(C),D,E(F,G)))	 <pre> graph TD     A((A)) --- B((B))     A --- D((D))     A --- E((E))     B --- C((C))     E --- F((F))     E --- G((G))         </pre>



## Exercice 5.5 Listes imbriquées (3)

```
#include <iostream>
#include <sstream>
using namespace std;

struct Noeud {
    string etiquette; Noeud* aine; Noeud* puine;
};

string read_label(istream& in) {
    const string separators = "(,)";
    ostringstream out; char c;
    while (in.get(c) and separators.find(c) == string::npos)
        out << c;
    in.putback(c);
    return out.str();
}

istream& operator>> (istream &in, Noeud& n) {
    auto m = n = new Noeud{etiquette : read_label(in)};
    char c;
    while (in.get(c)) { switch (c) {
        case ',': m->puine = new Noeud{etiquette : read_label(in)}; m = m->puine; break;
        case '(': in >> m->aine; break;
        case ')': return in;
        default : throw "bad expression";
    } }
    return in;
}

ostream& operator<<(ostream& out, Noeud* n) {
    if(n != nullptr) {
        out << n->etiquette << flush;
        if(Noeud* m = n->aine) {
            out << '(' << flush;
            while(m) {
                out << m << flush;
                m = m->puine;
                if(m) out << ',' << flush;
            }
            out << ')' << flush;
        }
    }
    return out;
}

int main() {
    for (string arbre: { "A(B(C,D(F,G),E),H(I,J))", "A(B,C(E,F(H,I)),D(G(J)))",
        "A(B(E,F),C(G,H,I))", "A(B,C(D,E(F,G,H)))" }) {
        Noeud* n;
        istringstream(arbre) >> n;
        cout << n << endl;
    }
}
```

## Exercice 5.6      Nomenclature (2)

```
size_t taille(const Noeud* n) {
    return n ? taille(n->aîne) + taille(n->puîne) + 1 : 0;
}

size_t hauteur(const Noeud* n) {
    return n ? max(hauteur(n->aîne)+1, hauteur(n->puîne)) : 0;
}

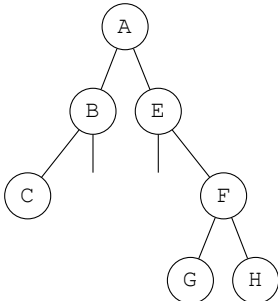
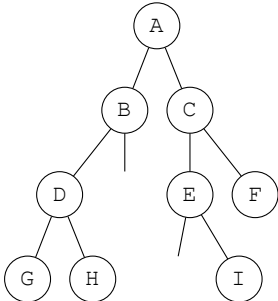
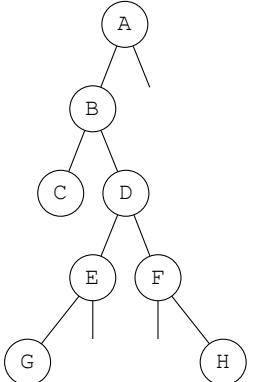
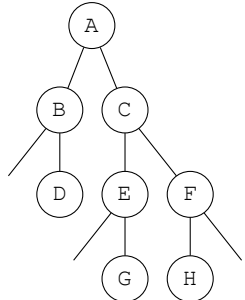
// .first = degre du noeud
// .second = degre de l'arbre
pair<size_t, size_t> degres(const Noeud* n) {
    if (n == nullptr)
        return {0, 0};
    auto dp = degres(n->puîne);
    auto da = degres(n->aîne);
    return {dp.first + 1, max({dp.first + 1, dp.second, da.second})};
}

size_t degre(const Noeud* n) {
    return degres(n).second;
}

string feuilles(const Noeud* n) {
    return (n) ? ( (n->aîne) ? feuilles(n->aîne) : n->etiquette + " " ) +
    feuilles(n->puîne) : "";
}

string internes(const Noeud* n) {
    return (n) ? ( (n->aîne) ? n->etiquette + " " + internes(n->aîne) : "" ) +
    internes(n->puîne) : "";
}
```

## Exercice 5.7 Parcours arbre binaire (1)

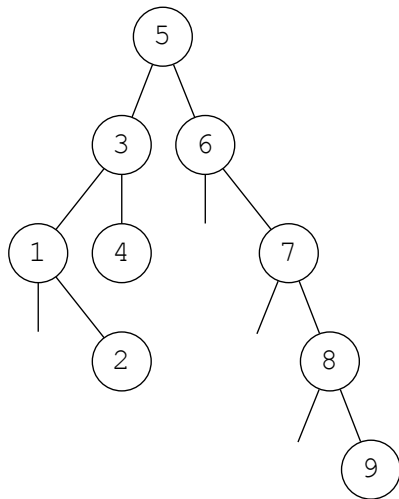
 <pre> graph TD     A((A)) --&gt; B((B))     A --&gt; E((E))     B --&gt; C((C))     E --&gt; F((F))     F --&gt; G((G))     F --&gt; H((H)) </pre>	<pre> pre : A B C E F G H sym : C B A E G F H post : C B G H F E A </pre>
 <pre> graph TD     A((A)) --&gt; B((B))     A --&gt; C((C))     B --&gt; D((D))     D --&gt; G((G))     D --&gt; H((H))     C --&gt; E((E))     E --&gt; I((I))     C --&gt; F((F)) </pre>	<pre> pre : A B D G H C E I F sym : G D H B A E I C F post : G H D B I E F C A </pre>
 <pre> graph TD     A((A)) --&gt; B((B))     B --&gt; C((C))     B --&gt; D((D))     D --&gt; E((E))     E --&gt; G((G))     D --&gt; F((F))     F --&gt; H((H)) </pre>	<pre> pre : A B C D E G F H sym : C B G E D F H A post : C G E H F D B A </pre>
 <pre> graph TD     A((A)) --&gt; B((B))     A --&gt; C((C))     B --&gt; D((D))     C --&gt; E((E))     E --&gt; G((G))     C --&gt; F((F))     F --&gt; H((H)) </pre>	<pre> pre : A B D C E G F H sym : B D A E G C H F post : D B G E H F C A </pre>

**Exercice 5.8      Parcours arbre binaire (2)**

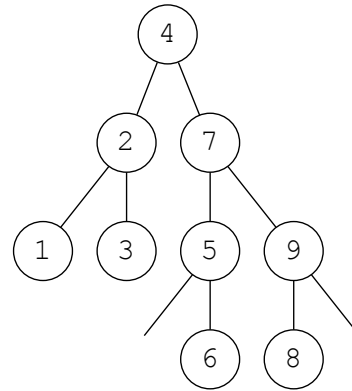
<pre>pre  : A B D H I E K C F G sym  : H D I B K E A C G F</pre>	<pre>graph TD     A((A)) --&gt; B((B))     A --&gt; C((C))     B --&gt; D((D))     B --&gt; E((E))     D --&gt; H((H))     D --&gt; I((I))     E --&gt; K((K))     C --&gt; F((F))     F --&gt; G((G))</pre>
<pre>sym  : F G A E C D B H post : G F A D C H B E</pre>	<pre>graph TD     E((E)) --&gt; A((A))     E --&gt; B((B))     A --&gt; F((F))     F --&gt; G((G))     B --&gt; C((C))     B --&gt; H((H))     C --&gt; D((D))</pre>
<pre>pre  : G A B D I C F H E sym  : B A D I G H F C E</pre>	<pre>graph TD     G((G)) --&gt; A((A))     G --&gt; C((C))     A --&gt; B((B))     A --&gt; D((D))     D --&gt; I((I))     C --&gt; F((F))     C --&gt; E((E))     F --&gt; H((H))</pre>
<pre>sym  : A G D B E F H C post : A D E B G C H F</pre>	<pre>graph TD     F((F)) --&gt; G((G))     F --&gt; H((H))     G --&gt; A((A))     G --&gt; B((B))     B --&gt; D((D))     B --&gt; E((E))     H --&gt; C((C))</pre>

## Exercice 5.9 Insertions dans un arbre binaire de recherche

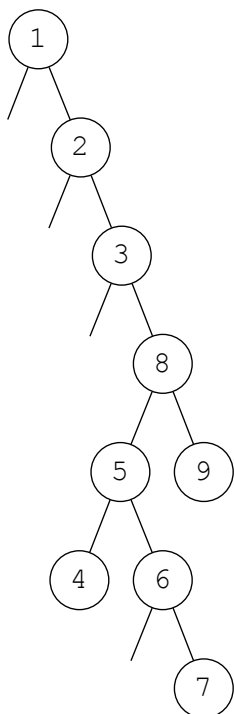
5, 3, 1, 6, 7, 4, 8, 2, 9



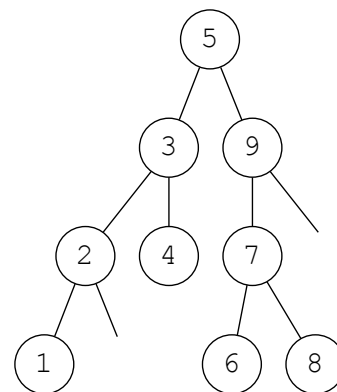
4, 7, 2, 1, 5, 9, 6, 8, 3



1, 2, 3, 8, 5, 6, 4, 7, 9

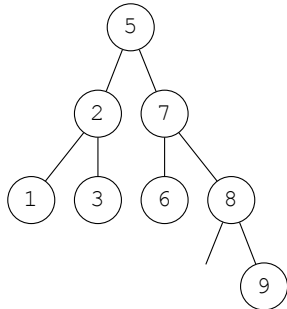


5, 3, 9, 7, 6, 8, 2, 4, 1

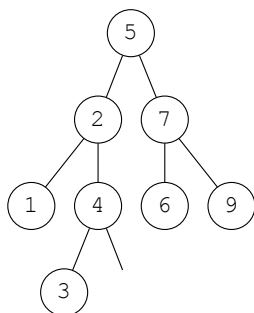


## Exercice 5.10 Suppressions d'un arbre binaire de recherche

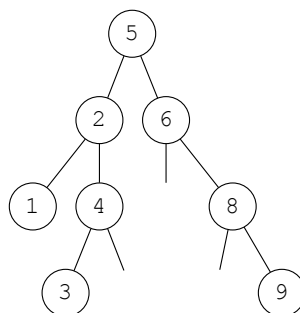
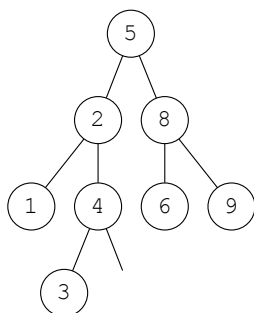
- Supprimer la clé 4



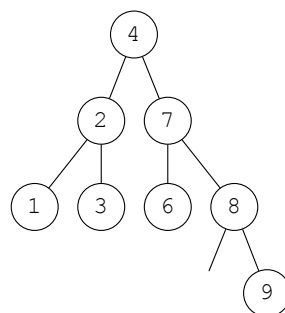
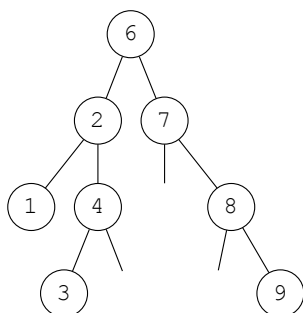
- Supprimer la clé 8



- Supprimer la clé 7 (Hibbard, 2 solutions)



- Supprimer la clé 5 (Hibbard, 2 solutions)

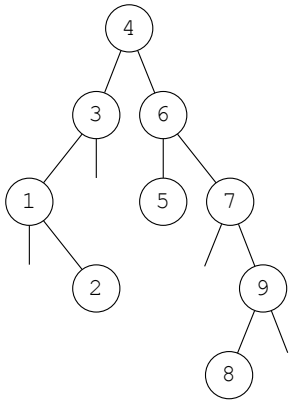
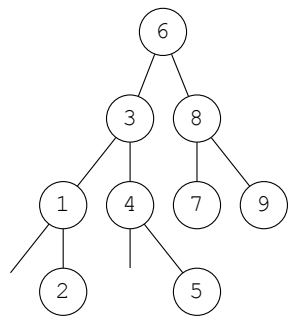
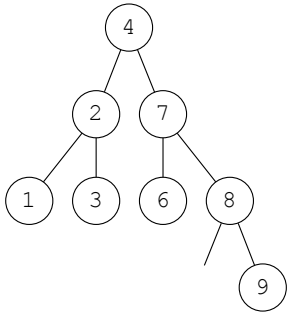


## Exercice 5.11 Parcours d'un arbre binaire de recherche

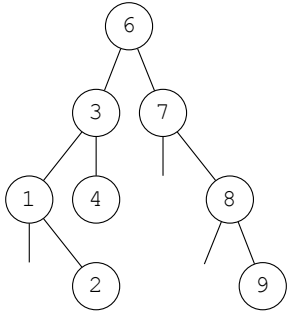
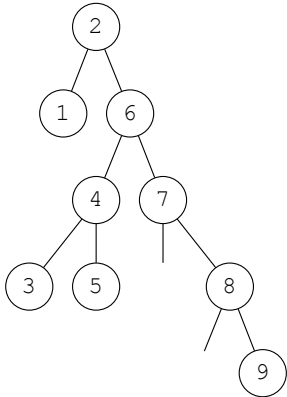
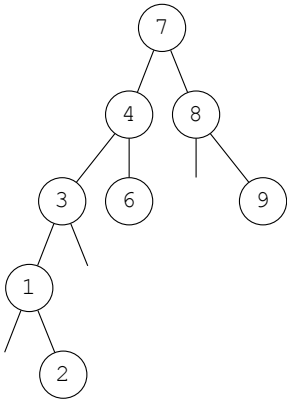
Notons que l'on connaît aussi le parcours symétrique de l'arbre, qui est le parcours ordonné par clé croissante.

<p>Pré-ordre : 4 3 1 2 6 5 7 9 8</p>	<p>Post-ordre : 3 2 4 6 8 9 7 5 1</p>
<p>Pré-ordre : 6 3 1 2 4 7 8 9</p>	<p>Post-ordre: 2 1 3 6 4 9 8 7</p>
<p>Pré-ordre : 4 2 1 3 5 6</p>	<p>Post-ordre : 1 2 4 6 8 7 5 3</p>

## Exercice 5.12    Equilibre

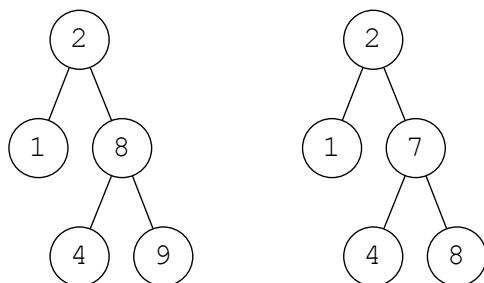
 <pre> graph TD     4((4)) --- 3((3))     4 --- 6((6))     3 --- 1((1))     3 --- 5((5))     1 --- 2((2))     6 --- 5     6 --- 7((7))     7 --- 9((9))     7 --- 8((8))         </pre>	<p>Les noeuds 3, 6, et 7 sont déséquilibrés</p>
 <pre> graph TD     6((6)) --- 3((3))     6 --- 8((8))     3 --- 1((1))     3 --- 4((4))     1 --- 2((2))     4 --- 5((5))     8 --- 7((7))     8 --- 9((9))         </pre>	<p>L'arbre est équilibré</p>
 <pre> graph TD     4((4)) --- 2((2))     4 --- 7((7))     2 --- 1((1))     2 --- 3((3))     7 --- 6((6))     7 --- 8((8))     8 --- 9((9))         </pre>	<p>L'arbre est équilibré</p>



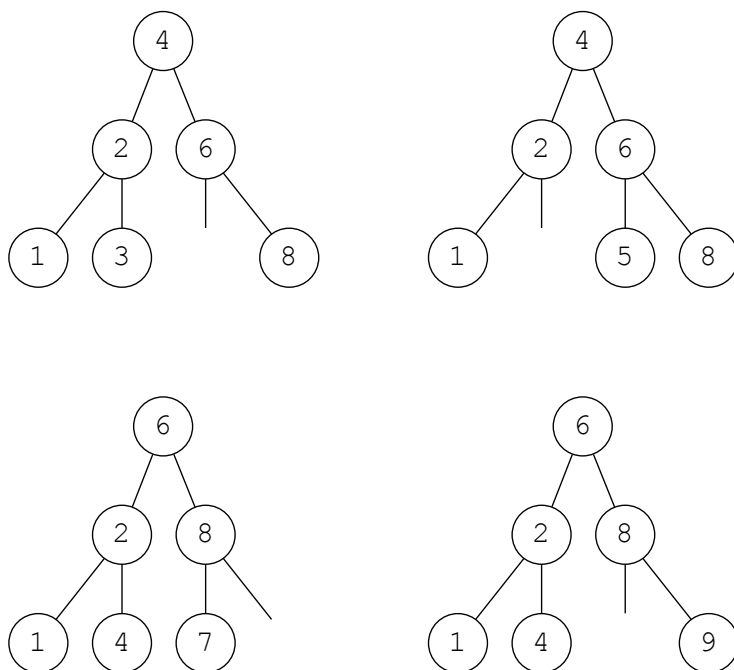
 <pre> graph TD     6((6)) --- 3((3))     6 --- 7((7))     3 --- 1((1))     3 --- 4((4))     1 --- 2((2))     7 --- 8((8))     8 --- 9((9))         </pre>	<p>Le nœud 7 est déséquilibré</p>
 <pre> graph TD     2((2)) --- 1((1))     2 --- 6((6))     6 --- 4((4))     6 --- 7((7))     4 --- 3((3))     4 --- 5((5))     7 --- 8((8))     8 --- 9((9))         </pre>	<p>Les nœuds 2 et 7 sont déséquilibrés</p>
 <pre> graph TD     7((7)) --- 4((4))     7 --- 8((8))     4 --- 3((3))     4 --- 6((6))     3 --- 1((1))     1 --- 2((2))     8 --- 9((9))         </pre>	<p>Les nœuds 3, 4, et 7 sont déséquilibrés</p>

## Exercice 5.13 Insertion dans un arbre AVL

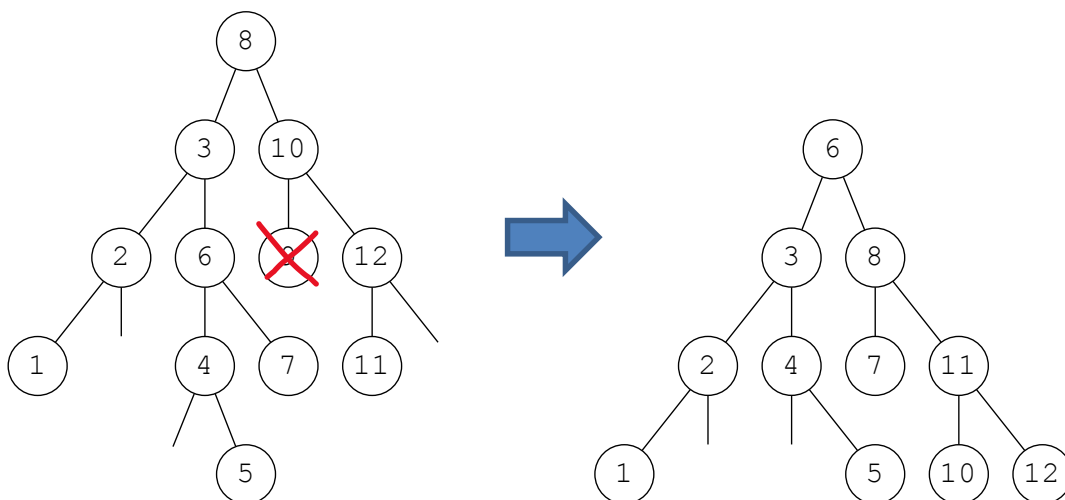
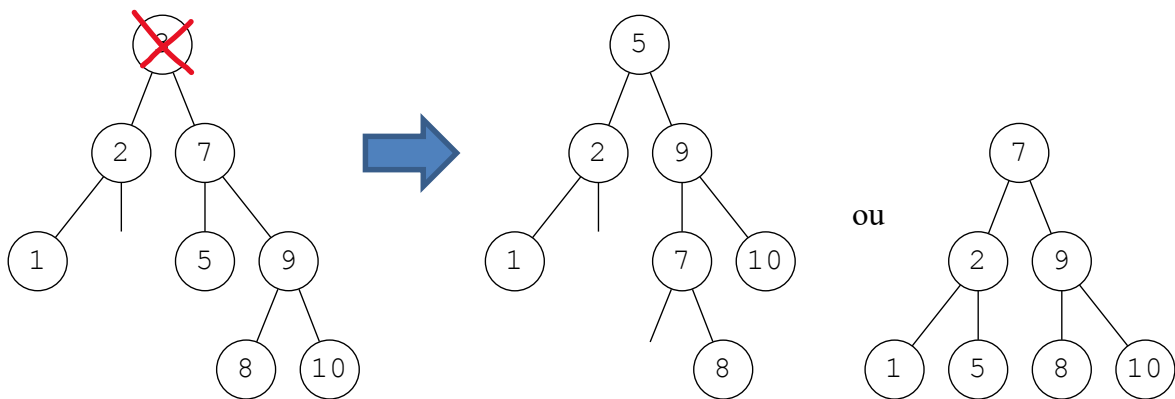
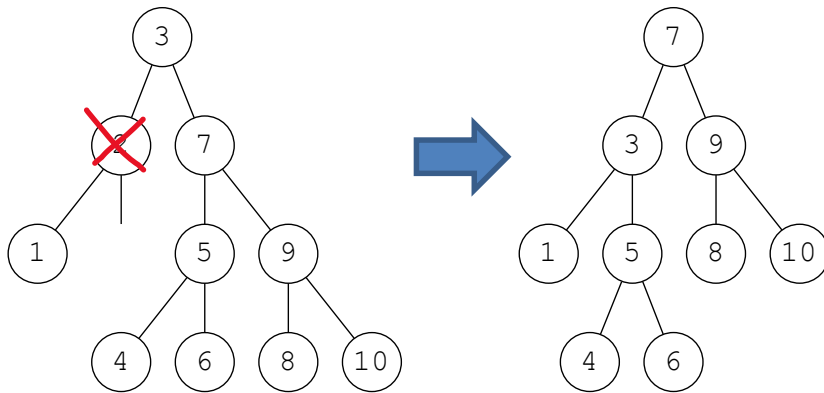
1.



2.



## Exercice 5.14 Suppression dans un arbre AVL

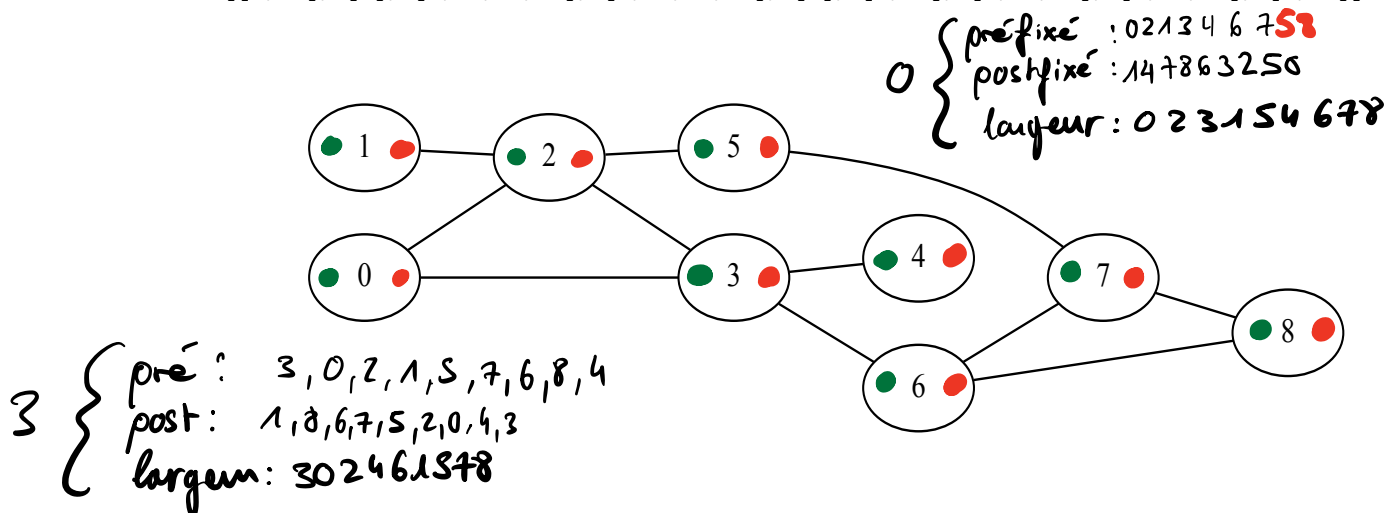


## Chapitre 6 : Graphes

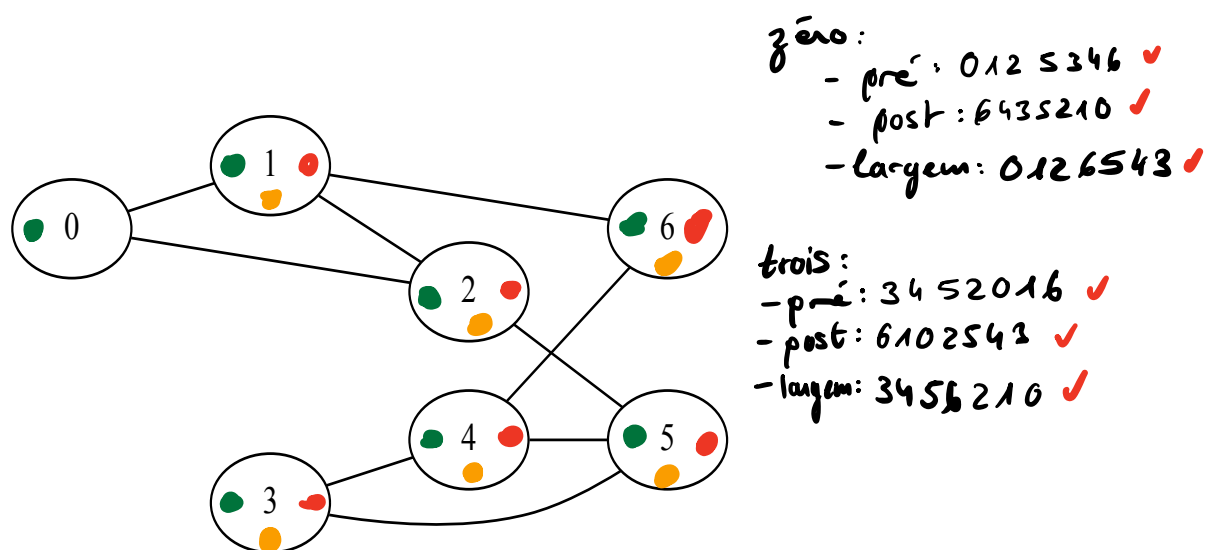
### Exercice 6.1 Parcours

Soit les graphes suivants dont les listes d'adjacence sont triées par ordre de sommet croissant.  
Donnez les parcours en profondeur pré et post-ordonnés ainsi que le parcours en largeur à partir des sommets 0 et 3.

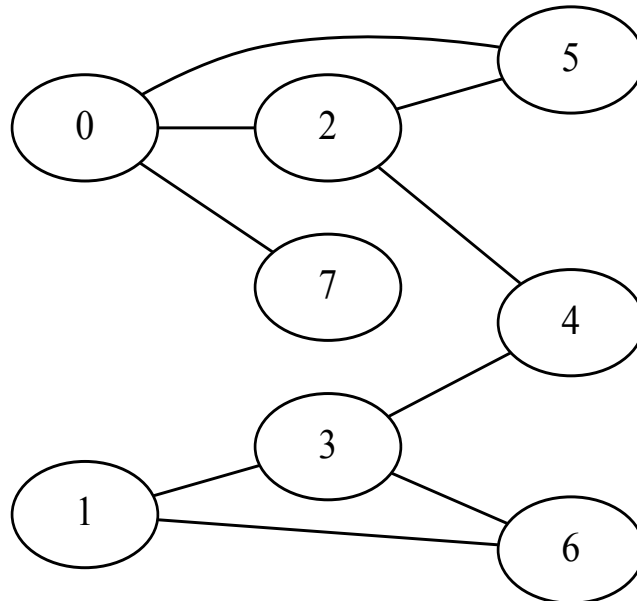
1. [[2, 3], [2], [0, 1, 3, 5], [0, 2, 4, 6], [3], [2, 7], [3, 7, 8], [5, 6, 8], [6, 7]]



2. [[1, 2], [0, 2, 6], [0, 1, 5], [4, 5], [3, 5, 6], [2, 3, 4], [1, 4]]

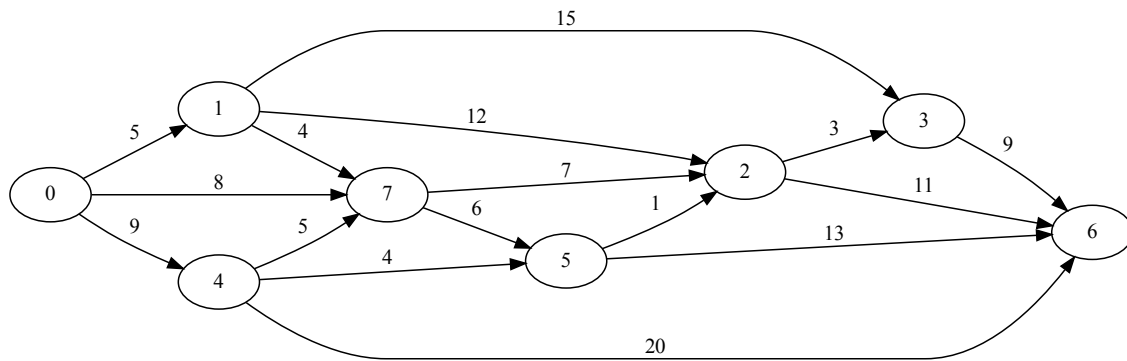


3.  $[[2, 5, 7], [3, 6], [0, 4, 5], [1, 4, 6], [2, 3], [0, 2], [1, 3], [0]]$



## Exercice 6.2 Dijkstra

Appliquez l'algorithme de Dijkstra aux graphes suivants en partant du sommet 0. Indiquez l'état des tableaux de distance et de dernier arc (ou parent) après chaque étape de relaxation de tous les sommets sortant d'un sommet, comme dans l'exemple suivant.



Après traitement du sommet 0

DistTo = [0, 5, inf, inf, 9, inf, inf, 8]

Parent = [None, 0, None, None, 0, None, None, 0]

Après traitement du sommet 1

DistTo = [0, 5, 17, 20, 9, inf, inf, 8]

Parent = [None, 0, 1, 1, 0, None, None, 0]

Après traitement du sommet 7

DistTo = [0, 5, 15, 20, 9, 14, inf, 8]

Parent = [None, 0, 7, 1, 0, 7, None, 0]

Après traitement du sommet 4

DistTo = [0, 5, 15, 20, 9, 13, 29, 8]

Parent = [None, 0, 7, 1, 0, 4, 4, 0]

Après traitement du sommet 5

DistTo = [0, 5, 14, 20, 9, 13, 26, 8]

Parent = [None, 0, 5, 1, 0, 4, 5, 0]

Après traitement du sommet 2

DistTo = [0, 5, 14, 17, 9, 13, 25, 8]

Parent = [None, 0, 5, 2, 0, 4, 2, 0]

Après traitement du sommet 3

DistTo = [0, 5, 14, 17, 9, 13, 25, 8]

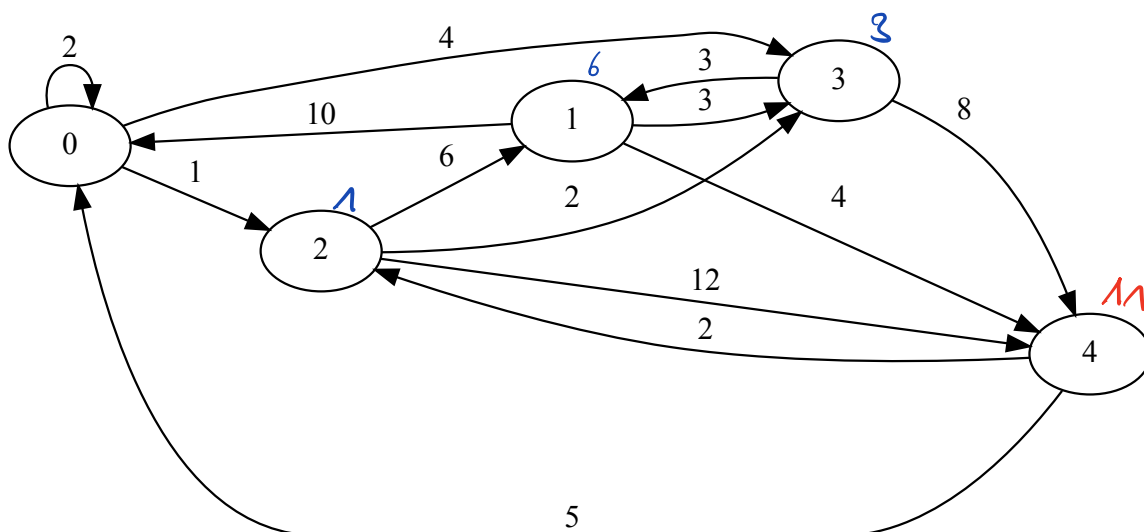
Parent = [None, 0, 5, 2, 0, 4, 2, 0]

Après traitement du sommet 6

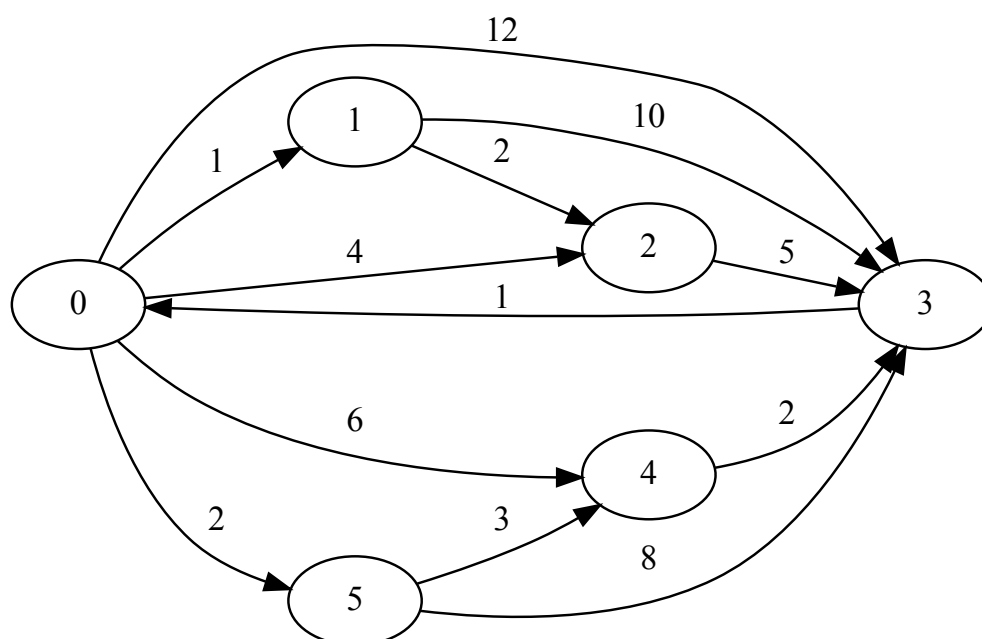
DistTo = [0, 5, 14, 17, 9, 13, 25, 8]

Parent = [None, 0, 5, 2, 0, 4, 2, 0]

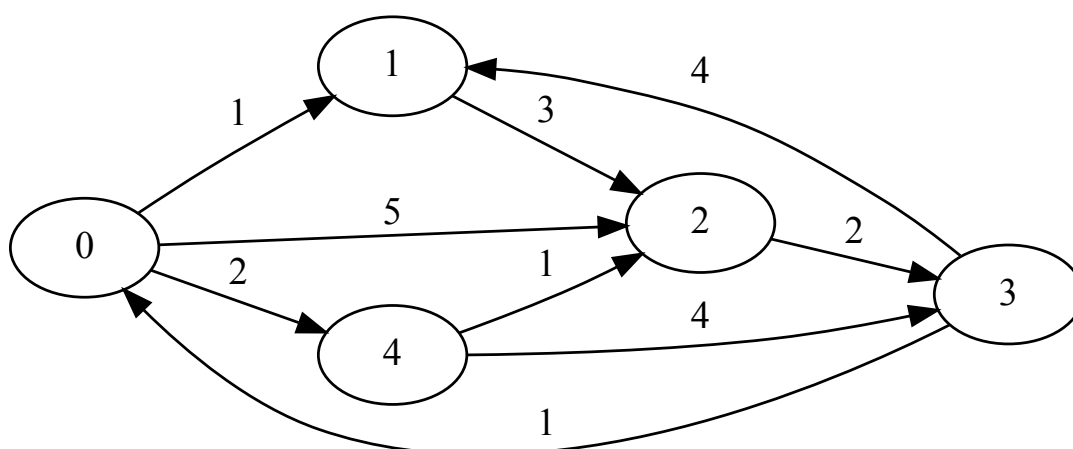
**a.**



**b.**



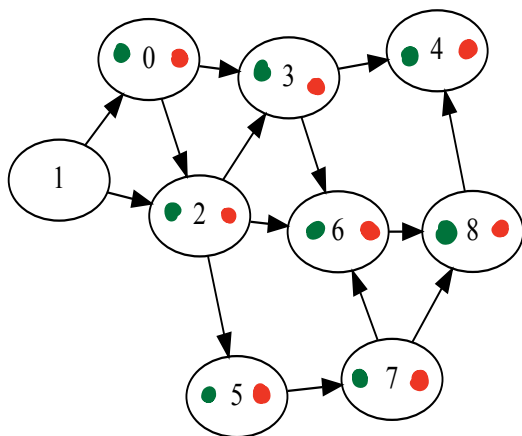
**c.**



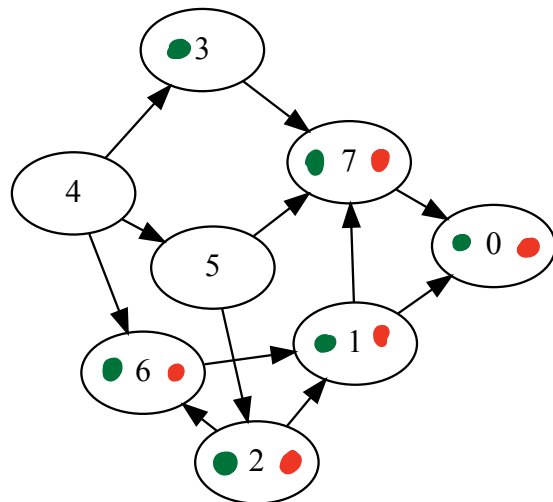
### Exercice 6.3 Tri topologique

Effectuer le tri topologique des graphes orientés suivants. Les listes de successeurs sont supposées triées par ordre croissant.

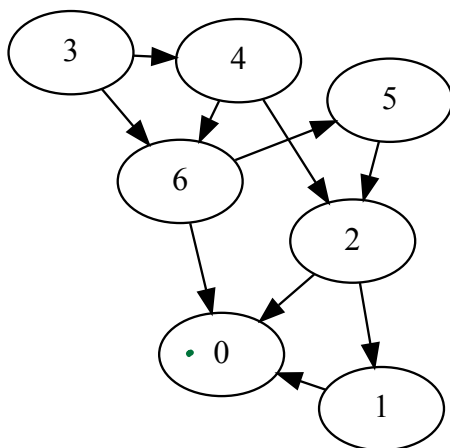
a.



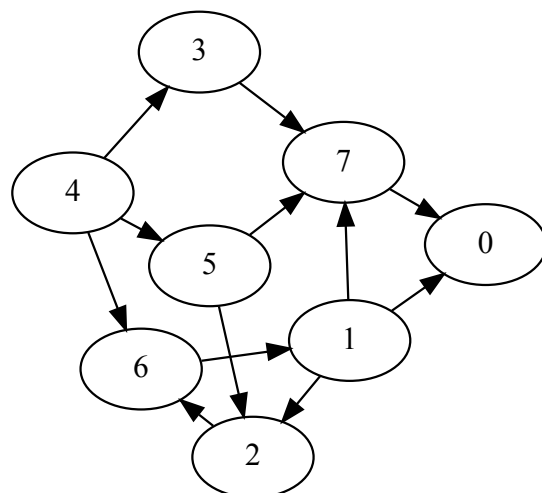
b.



c.



d.





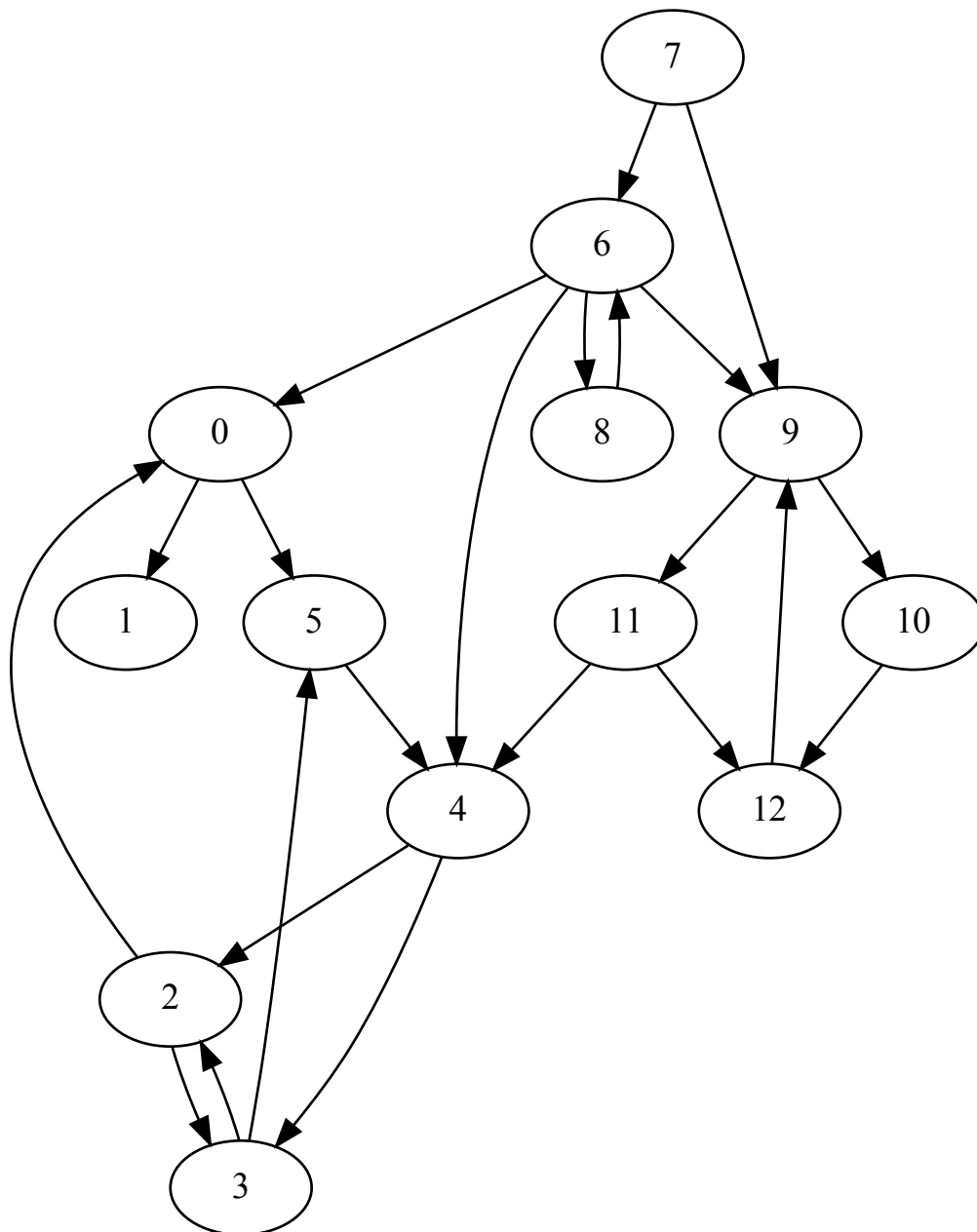
### Exercice 6.4 Composantes fortement connexes

Appliquez l'algorithme de Kosaraju-Sharir pour déterminer les composantes fortement connexes des graphes. Suivants. Indiquez pour chaque graphe

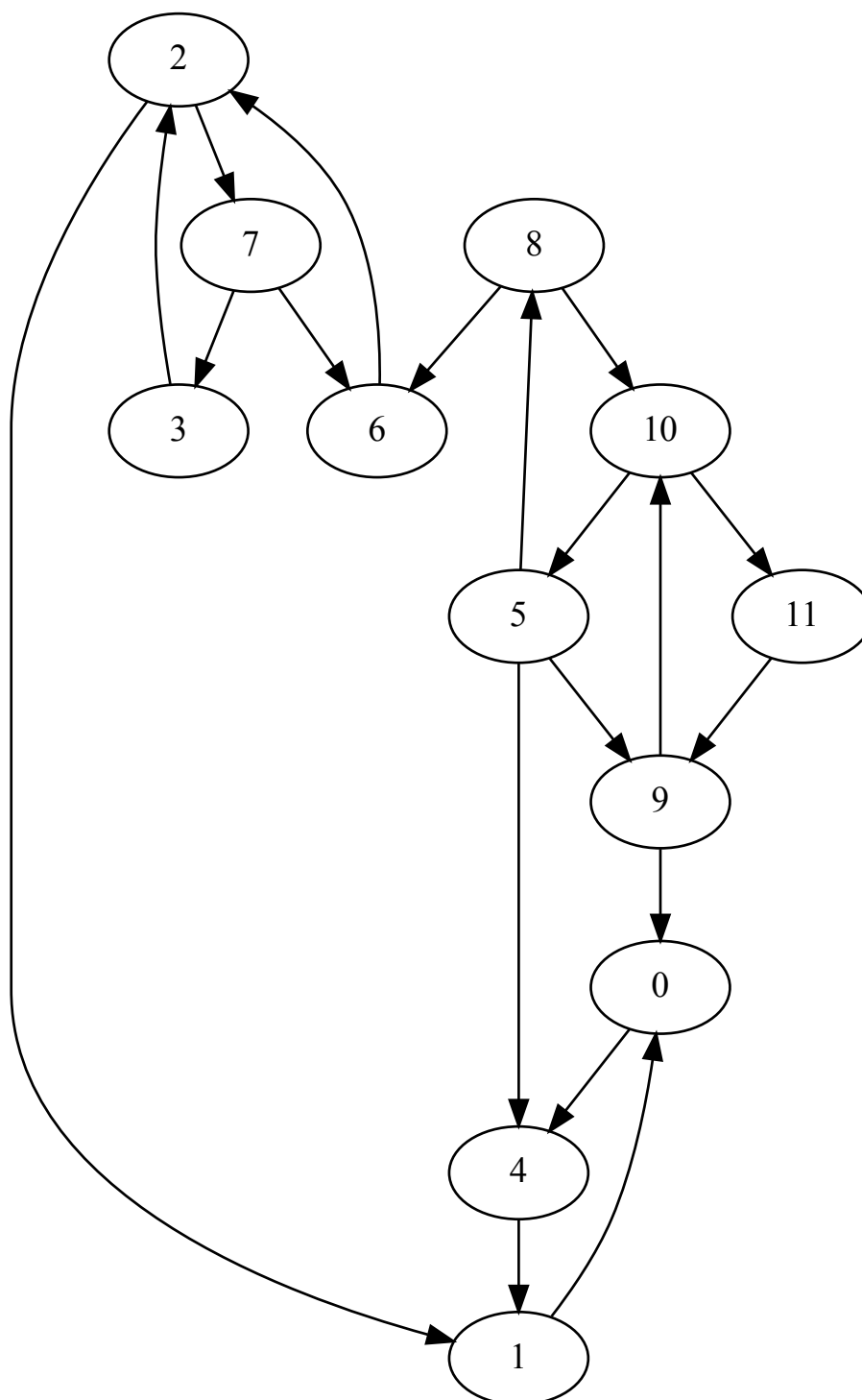
- Le parcours post-ordonné inverse du graphe inversé
- Les composantes fortement connexes du graphe, dont les sommets sont ordonnés selon l'ordre du parcours ayant permis de les déterminer.

On suppose que les listes d'adjacence du graphe sont ordonnées par indice de sommet croissant.

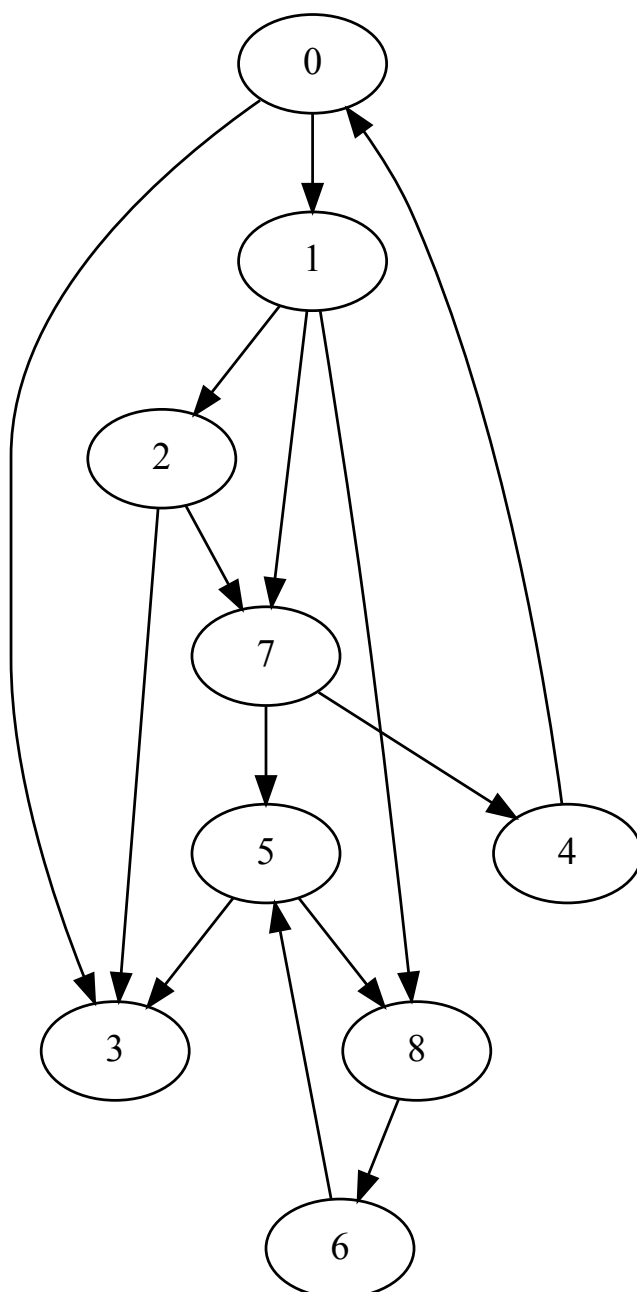
a.



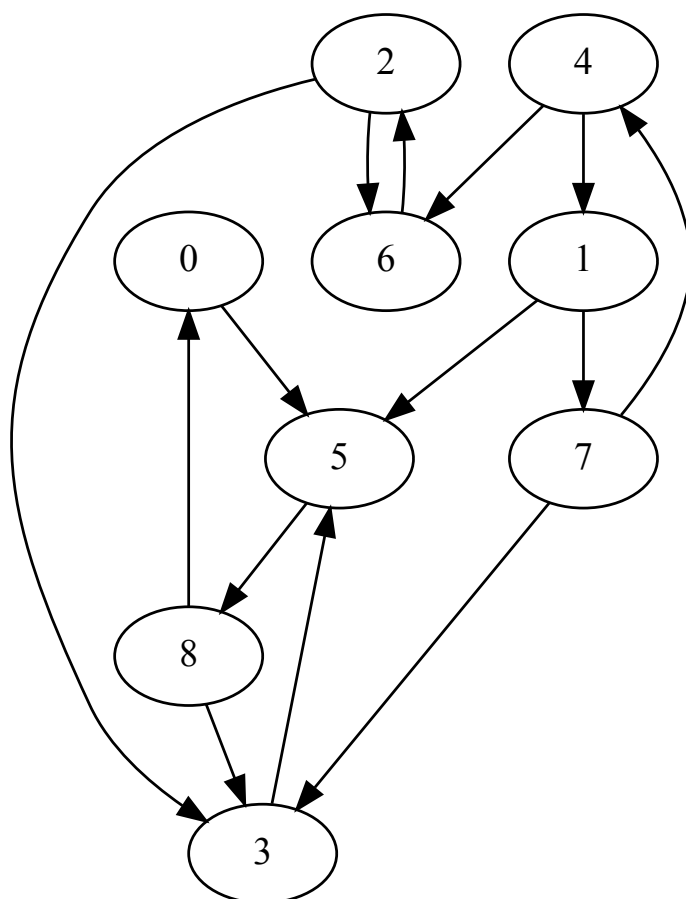
b.



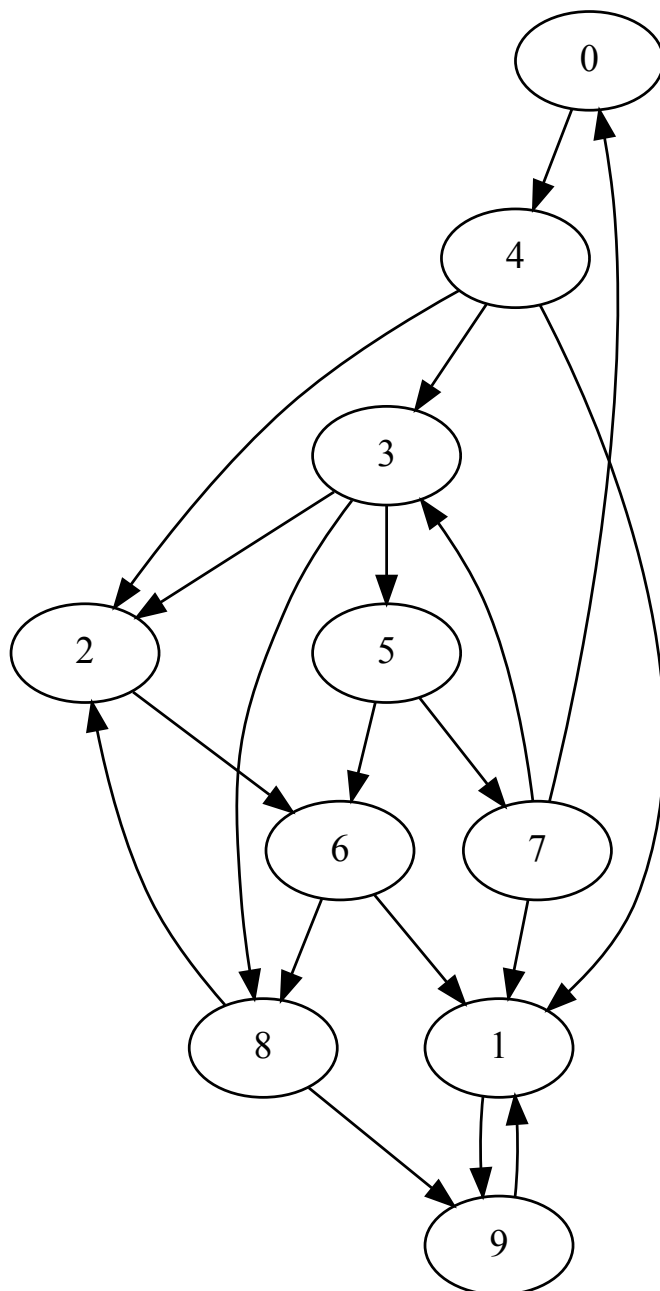
c.



d.



e.



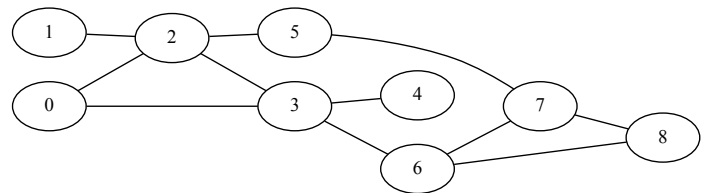
## Solutions

### Exercice 6.1 Parcours

pré-ordre : 0 2 1 3 4 6 7 5 8

post-ordre : 1 4 5 8 7 6 3 2 0

largeur : 0 2 3 1 5 4 6 7 8



Le parcours en profondeur est le suivant, en

notant ✓ quand on entre dans la fonction

parcourant un sommet, **X** quand on considère un sommet voisin qu'on ne fait pas l'appel récursif car il est déjà marqué, et **o** quand on sort de la fonction traitant le sommet. Le pré-ordre correspond aux symboles ✓, le post-ordre aux symboles **o**.

```

0 ✓ [2, 3]
| 2 ✓ [0, 1, 3, 5]
| | 0 X
| | 1 ✓ [2]
| | | 2 X
| | 1 o
| | 3 ✓ [0, 2, 4, 6]
| | | 0 X
| | | 2 X
| | | 4 ✓ [3]
| | | | 3 X
| | | 4 o
| | | 6 ✓ [3, 7, 8]
| | | | 3 X
| | | | 7 ✓ [5, 6, 8]
| | | | | 5 ✓ [2, 7]
| | | | | | 2 X
| | | | | 7 X
| | | | 5 o
| | | | 6 X
| | | | 8 ✓ [6, 7]
| | | | | 6 X
| | | | | 7 X
| | | | 8 o
| | | 7 o
| | | 8 X
| | 6 o
| | 3 o
| | 5 X
| 2 o
| 3 X
0 o
  
```

Le parcours en largeur est le suivant, en notant le contenu de la file entre crochets, suivis du sommet qui en sort (pop) et de sa liste de d'adjacence, puis des sommets qui sont insérés (push) avec un symbole ✓ si l'insertion a lieu et ✗ si ce n'est pas le cas, le sommet ayant déjà été marqué comme visité.

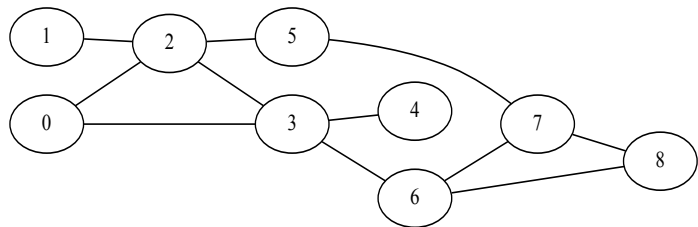
```
[0]
  pop : 0 [2, 3]
  push: 2 ✓ 3 ✓
[2, 3]
  pop : 2 [0, 1, 3, 5]
  push: 0 ✗ 1 ✓ 3 ✗ 5 ✓
[3, 1, 5]
  pop : 3 [0, 2, 4, 6]
  push: 0 ✗ 2 ✗ 4 ✓ 6 ✓
[1, 5, 4, 6]
  pop : 1 [2]
  push: 2 ✗
[5, 4, 6]
  pop : 5 [2, 7]
  push: 2 ✗ 7 ✓
[4, 6, 7]
  pop : 4 [3]
  push: 3 ✗
[6, 7]
  pop : 6 [3, 7, 8]
  push: 3 ✗ 7 ✗ 8 ✓
[7, 8]
  pop : 7 [5, 6, 8]
  push: 5 ✗ 6 ✗ 8 ✗
[8]
  pop : 8 [6, 7]
  push: 6 ✗ 7 ✗
```

**Depuis le sommet 3,**

pré-ordre : 3 0 2 1 5 7 6 8 4

post-ordre : 1 8 6 7 5 2 0 4 3

largeur : 3 0 2 4 6 1 5 7 8



Profondeur(3)	Largeur(3)
<pre> 3 ✓ [0, 2, 4, 6]   0 ✓ [2, 3]     2 ✓ [0, 1, 3, 5]       0 X       1 ✓ [2]         2 X         1 o       3 X       5 ✓ [2, 7]         2 X         7 ✓ [5, 6, 8]           5 X           6 ✓ [3, 7, 8]             3 X             7 X           8 ✓ [6, 7]             6 X             7 X             8 o         6 o         8 X       7 o     5 o   2 o   3 X   0 o   2 X   4 ✓ [3]     3 X     4 o     6 X 3 o </pre>	<pre> [3] pop : 3 [0, 2, 4, 6] push: 0 ✓ 2 ✓ 4 ✓ 6 ✓ [0, 2, 4, 6] pop : 0 [2, 3] push: 2 X 3 X [2, 4, 6] pop : 2 [0, 1, 3, 5] push: 0 X 1 ✓ 3 X 5 ✓ [4, 6, 1, 5] pop : 4 [3] push: 3 X [6, 1, 5] pop : 6 [3, 7, 8] push: 3 X 7 ✓ 8 ✓ [1, 5, 7, 8] pop : 1 [2] push: 2 X [5, 7, 8] pop : 5 [2, 7] push: 2 X 7 X [7, 8] pop : 7 [5, 6, 8] push: 5 X 6 X 8 X [8] pop : 8 [6, 7] push: 6 X 7 X </pre>

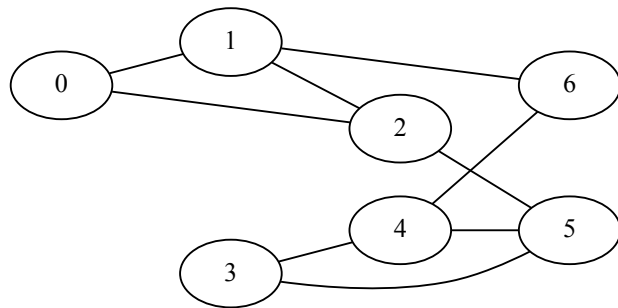


Depuis le sommet 0,

pré-ordre : 0 1 2 5 3 4 6

post-ordre : 6 4 3 5 2 1 0

largeur : 0 1 2 6 5 4 3



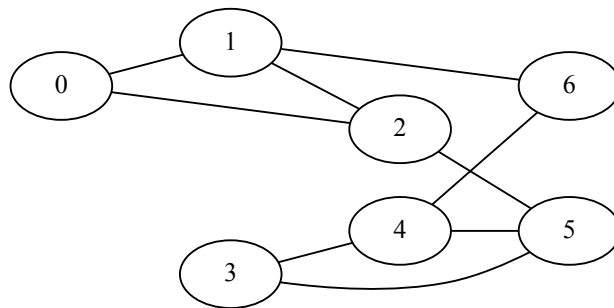
Profondeur(0)	Largeur(0)
0 ✓ [1, 2]	[0]
1 ✓ [0, 2, 6]	pop : 0 [1, 2]
0 X	push: 1 ✓ 2 ✓
2 ✓ [0, 1, 5]	[1, 2]
0 X	pop : 1 [0, 2, 6]
1 X	push: 0 X 2 X 6 ✓
5 ✓ [2, 3, 4]	[2, 6]
2 X	pop : 2 [0, 1, 5]
3 ✓ [4, 5]	push: 0 X 1 X 5 ✓
4 ✓ [3, 5, 6]	[6, 5]
3 X	pop : 6 [1, 4]
5 X	push: 1 X 4 ✓
6 ✓ [1, 4]	[5, 4]
1 X	pop : 5 [2, 3, 4]
4 X	push: 2 X 3 ✓ 4 X
6 o	[4, 3]
4 o	pop : 4 [3, 5, 6]
5 X	push: 3 X 5 X 6 X
3 o	[3]
4 X	pop : 3 [4, 5]
5 o	push: 4 X 5 X
2 o	
6 X	
1 o	
2 X	
0 o	

**Depuis le sommet 3,**

pré-ordre : 3 4 5 2 0 1 6

post-ordre : 6 1 0 2 5 4 3

largeur : 3 4 5 6 2 1 0



Profondeur(3)	Largeur(3)
3 ✓ [4, 5]   4 ✓ [3, 5, 6]     3 X     5 ✓ [2, 3, 4]       2 ✓ [0, 1, 5]         0 ✓ [1, 2]           1 ✓ [0, 2, 6]           0 X           2 X           6 ✓ [1, 4]             1 X             4 X           6 o         1 o       2 X       0 o     1 X     5 X     2 o     3 X     4 X     5 o     6 X   4 o   5 X 3 o	[3] pop : 3 [4, 5] push: 4 ✓ 5 ✓ [4, 5] pop : 4 [3, 5, 6] push: 3 X 5 X 6 ✓ [5, 6] pop : 5 [2, 3, 4] push: 2 ✓ 3 X 4 X [6, 2] pop : 6 [1, 4] push: 1 ✓ 4 X [2, 1] pop : 2 [0, 1, 5] push: 0 ✓ 1 X 5 X [1, 0] pop : 1 [0, 2, 6] push: 0 X 2 X 6 X [0] pop : 0 [1, 2] push: 1 X 2 X

## Exercice 6.2 Dijkstra

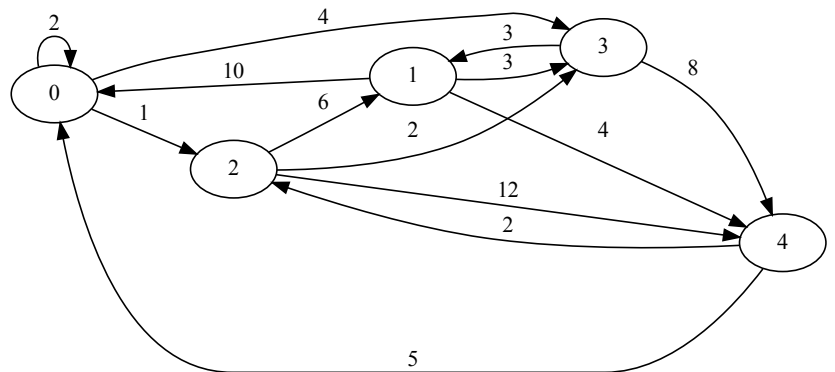
- a. Après traitement du sommet 0  
DistTo = [0, inf, 1, 4, inf]  
Parent = [None, None, 0, 0, None]

Après traitement du sommet 2  
DistTo = [0, 7, 1, 3, 13]  
Parent = [None, 2, 0, 2, 2]

Après traitement du sommet 3  
DistTo = [0, 6, 1, 3, 11]  
Parent = [None, 3, 0, 2, 3]

Après traitement du sommet 1  
DistTo = [0, 6, 1, 3, 10]  
Parent = [None, 3, 0, 2, 1]

Après traitement du sommet 4  
DistTo = [0, 6, 1, 3, 10]  
Parent = [None, 3, 0, 2, 1]



- b. Après traitement du sommet 0  
DistTo = [0, 1, 4, 12, 6, 2]  
Parent = [None, 0, 0, 0, 0, 0]

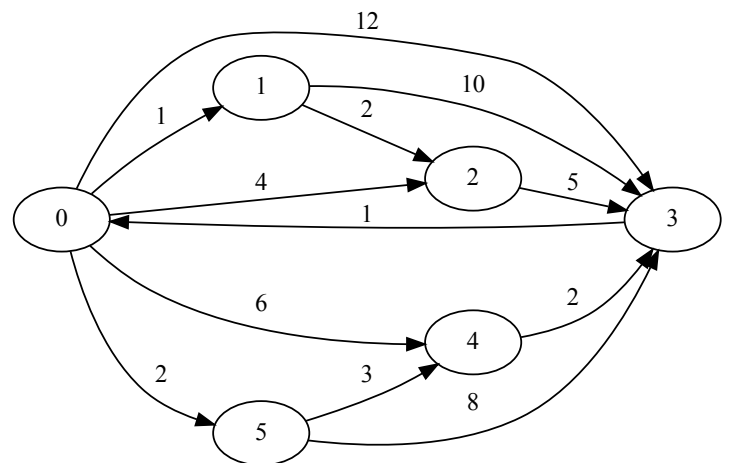
Après traitement du sommet 1  
DistTo = [0, 1, 3, 11, 6, 2]  
Parent = [None, 0, 1, 1, 0, 0]

Après traitement du sommet 5  
DistTo = [0, 1, 3, 10, 5, 2]  
Parent = [None, 0, 1, 5, 5, 0]

Après traitement du sommet 2  
DistTo = [0, 1, 3, 8, 5, 2]  
Parent = [None, 0, 1, 2, 5, 0]

Après traitement du sommet 4  
DistTo = [0, 1, 3, 7, 5, 2]  
Parent = [None, 0, 1, 4, 5, 0]

Après traitement du sommet 3  
DistTo = [0, 1, 3, 7, 5, 2]  
Parent = [None, 0, 1, 4, 5, 0]



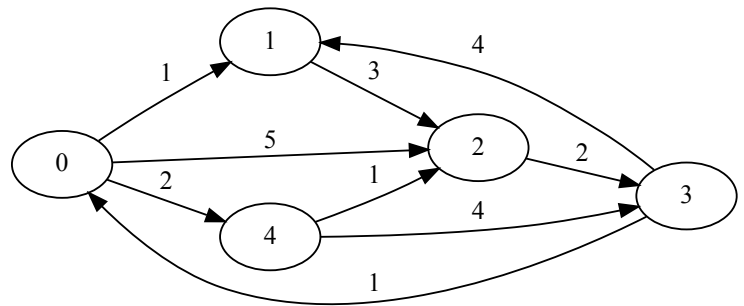
- c. Après traitement du sommet 0  
 DistTo = [0, 1, 5, inf, 2]  
 Parent = [None, 0, 0, None, 0]

Après traitement du sommet 1  
 DistTo = [0, 1, 4, inf, 2]  
 Parent = [None, 0, 1, None, 0]

Après traitement du sommet 4  
 DistTo = [0, 1, 3, 6, 2]  
 Parent = [None, 0, 4, 4, 0]

Après traitement du sommet 2  
 DistTo = [0, 1, 3, 5, 2]  
 Parent = [None, 0, 4, 2, 0]

Après traitement du sommet 3  
 DistTo = [0, 1, 3, 5, 2]  
 Parent = [None, 0, 4, 2, 0]



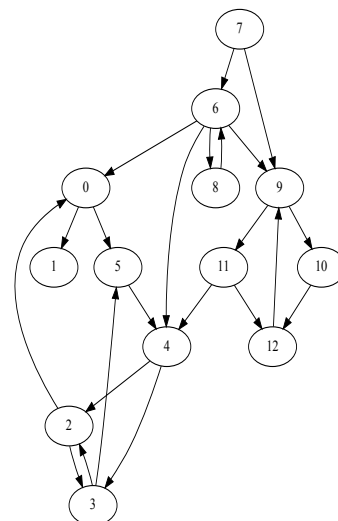
### Exercice 6.3 Tri topologique

- [1, 0, 2, 5, 7, 3, 6, 8, 4]
- [4, 3, 5, 2, 6, 1, 7, 0]
- [3, 4, 6, 5, 2, 1, 0]
- Impossible à trier en raison du cycle 1,2,6,1

### Exercice 6.4 Composantes fortement connexes

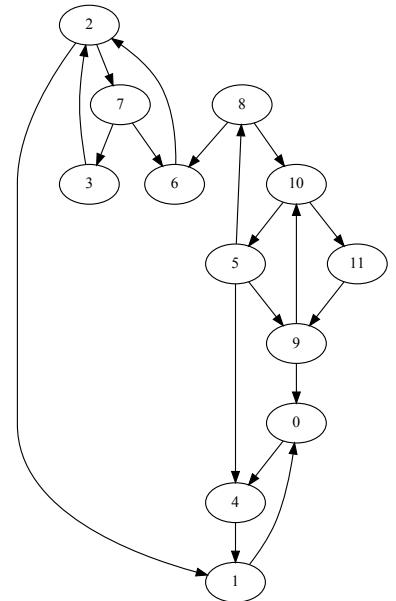
- a. Post-ordre inverse du graphe inversé:  
 [1, 0, 2, 3, 4, 11, 9, 12, 10, 6, 8, 7, 5]

CC(0): 1  
 CC(1): 0 5 4 2 3  
 CC(2): 11 12 9 10  
 CC(3): 6 8  
 CC(4): 7



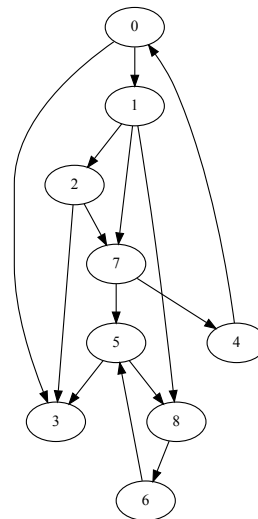
- b. Post-ordre inverse du graphe inversé:  
[0, 1, 4, 2, 6, 8, 5, 10, 9, 11, 3, 7]

CC(0): 0 4 1  
CC(1): 2 7 3 6  
CC(2): 8 10 5 9 11



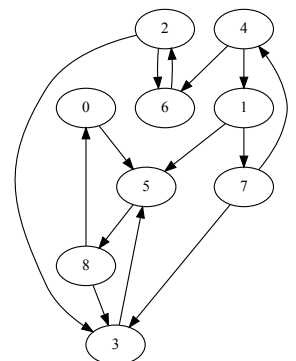
- c. Post-ordre inverse du graphe inversé:  
[3, 5, 6, 8, 0, 4, 7, 2, 1]

CC(0): 3  
CC(1): 5 8 6  
CC(2): 0 1 2 7 4



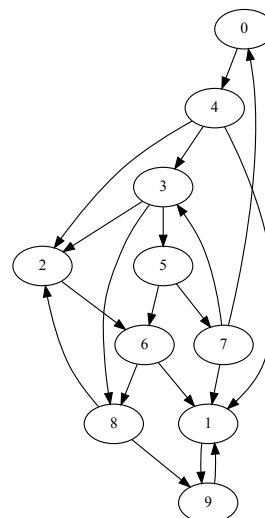
- d. Post-ordre inverse du graphe inversé:  
[0, 8, 5, 3, 2, 6, 1, 4, 7]

CC(0): 0 5 8 3  
CC(1): 2 6  
CC(2): 1 7 4



- e. Post-ordre inverse du graphe inversé:  
[1, 9, 6, 2, 8, 0, 7, 5, 3, 4]

CC(0): 1 9  
CC(1): 6 8 2  
CC(2): 0 4 3 5 7



## Chapitre 7 : Tables de hachage

### Exercice 7.1      *Résolution de collisions par chaînage*

Soit une table de hachage avec résolution des collisions par chaînage où on stocke des entiers avec

- M : le nombre de listes simplement chaînées
- N : le nombre d'entiers stockés.

L'insertion d'un entier déjà présent écrase la valeur. Pour garder la longueur moyenne d'une liste  $N / M$  constante, on applique la stratégie suivante.

- Doubler la taille du tableau quand  $N / M \geq 8$  lors de l'insertion
- Réduire le tableau à la moitié quand  $N / M \leq 2$  lors de la suppression
- On doit re-hacher toutes les clés quand on change la taille
- Parcours, insertion et suppression des éléments dans les listes simplement chaînées se font dans le sens ou du côté les plus efficace. L'insertion se fait après le re-hachage quand celui-ci a lieu

Fonction de hachage :  $h(k) = k \% M$

Donner l'état de la table de hachage et les valeurs de N et de M en effectuant les opérations suivantes :

- Initialisation à  $N = 0$  et  $M = 4$
- Insérer 2, 4, 5, 1, 0, 2, 3
- Suppression de 2

### Exercice 7.2      *Résolution de collisions par chaînage (2)*

Même exercice que le précédent, mais en effectuant les opérations suivantes :

- Initialisation à  $N = 0$  et  $M = 2$
- Insérer 2, 5, 3, 7, 4, 6
- Supprimer 3, 6
- Insérer 9, 0, 1, 3, 8

### **Exercice 7.3      Résolution de collisions par sondage linéaire**

Soit une table de hachage avec résolution des collisions par sondage linéaire où on stocke des entiers avec

- M : la taille de la table de hachage
- N : le nombre d'entiers stockés.

L'insertion d'un entier déjà présent écrase la valeur. Pour garder le taux d'occupation moyen  $N / M \leq 1/2$  :

- Doubler la taille du tableau quand l'insertion entraîne que  $N / M \geq 1/2$
- Réduire le tableau à la moitié quand la suppression entraîne que  $N / M \leq 1/8$

Fonction de hachage :  $h(k) = k \% 13 \% M$

Donner l'état de la table de hachage et les valeurs de N et de M en effectuant les opérations suivantes :

- Initialisation à  $N = 0$  et  $M = 4$
- Insertion de 5, 2, 3, 15, 8, 28
- Suppression de 2, 3, 15, 5

## Solutions

### Exercice 7.1 Résolution de collisions par chaînage

Initialiser				
Insérer 2			2	
Insérer 4	4		2	
Insérer 5	4	5	2	
Insérer 1	4	1 → 5	2	
Insérer 0	0 → 4	1 → 5	2	
Insérer 2	0 → 4	1 → 5	2	
Insérer 3	0 → 4	1 → 5	2	3
Supprimer 2	4 → 0	3 → 5 → 1		

### Exercice 7.2 Résolution de collisions par chaînage (2)

Initialiser		
Insérer 2	2	
Insérer 5	2	5
Insérer 3	2	3 → 5
Insérer 7	2	7 → 3 → 5
Insérer 4	4 → 2	7 → 3 → 5
Insérer 6	6 → 4 → 2	7 → 3 → 5
Supprimer 3	6 → 4 → 2	7 → 5
Supprimer 6	5 → 7 → 2 → 4	
Insérer 9	9 → 5 → 7 → 4 → 2	
Insérer 0	0 → 9 → 5 → 7 → 4 → 2	
Insérer 1	1 → 0 → 9 → 5 → 7 → 4 → 2	
Insérer 3	2 → 4 → 0	3 → 7 → 5 → 9 → 1
Insérer 8	8 → 2 → 4 → 0	3 → 7 → 5 → 9 → 1



### Exercice 7.3 Résolution de collisions par sondage linéaire

- Initialisation à  $N = 0$  et  $M = 4$
- Insertion de 5, 2, 3, 15, 8, 28
- Suppression de 2, 3, 15, 5

Avec  $h(k) = k \% 13$ , on a  $h(2) = 2$ ,  $h(3) = 3$ ,  $h(5) = 5$ ,  $h(15) = 2$ ,  $h(8) = 8$ ,  $h(28) = 2$ , auquel il faut appliquer le modulo  $M$  en fonction de la taille du tableau.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	5														
		2			5										
		2	3		5										
		2	3	15	5										
		2	3	15	5			8							
		2	3	15	5	28		8							
		15	3	28	5			8							
		15	28		5			8							
		15	28					8							
		28			5			8							
8		28													

Ne pas oublier de rehacher les clés suivantes jusqu'à la première case vide lors des suppressions.