

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

Partie 3 : structures non linéaires

Version 0.3
25 mai 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

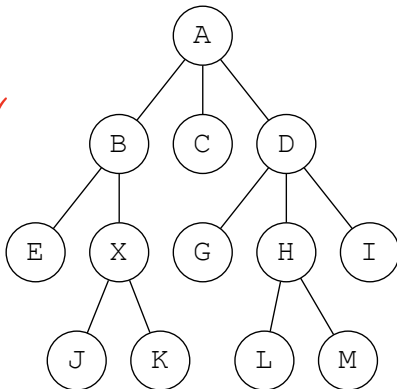
Chapitre 5 : Arbres

Exercice 5.1 Nomenclature

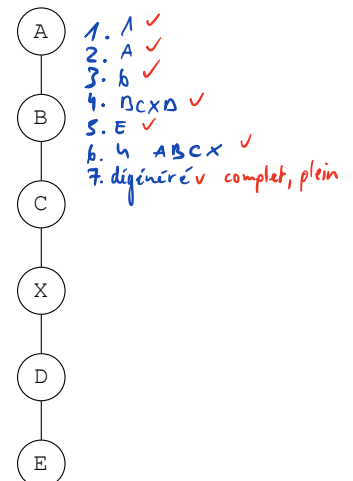
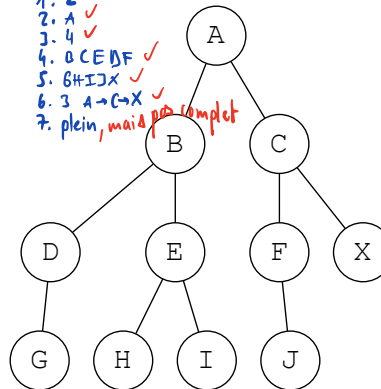
Pour chacun des arbres ci-dessous, indiquez

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

1. 3 ✓
2. A ✓
3. 4 ✓
4. A, B, C, D, E, G, H, I, J, K, L, M ✓
5. E, J, K, C, G, L, M, I ✓
6. 3 A → B → X ✓
7. rien ✓

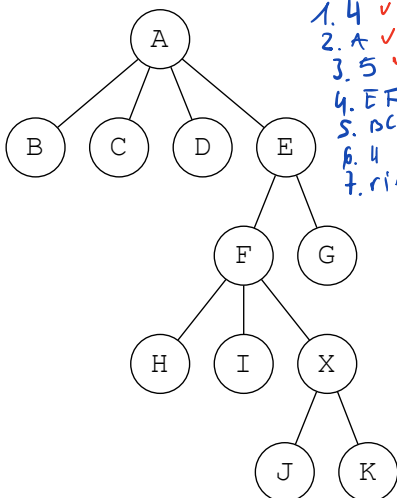


1. 2 ✓
2. A ✓
3. 4 ✓
4. A, C, E, D, F ✓
5. B, H, I, J, X ✓
6. 3 A → C → X ✓
7. plein, mais pas complet ✓

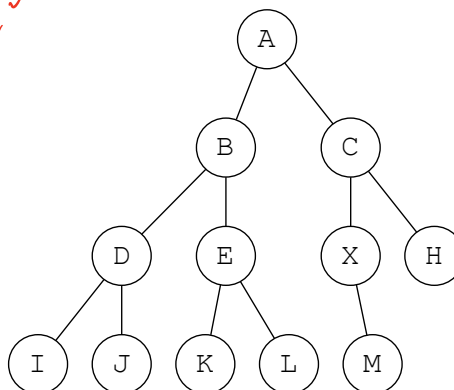


1. 1 ✓
2. A ✓
3. 6 ✓
4. A, B, C, X, D ✓
5. E ✓
6. 4 A → B → C → X ✓
7. dégénéré ✓ complet, plein

1. 4 ✓
2. A ✓
3. 5 ✓
4. E, F, X ✓
5. A, B, C, D, H, I, J, K, G ✓
6. 4 A → E → F → X ✓
7. rien ✓



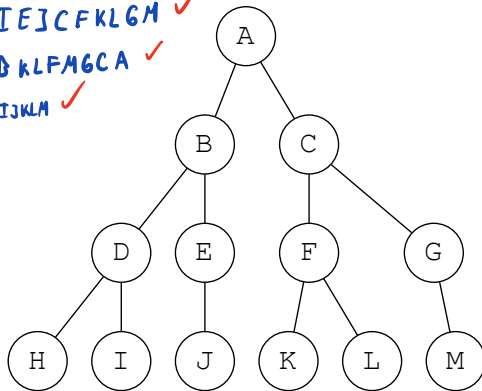
1. 2 ✓
2. A ✓
3. 4 ✓
4. A, B, C, D, E, X ✓
5. I, J, K, L, M, H ✓
6. 3 A → C → X ✓
7. plein ✓, complet, mais pas dégénéré



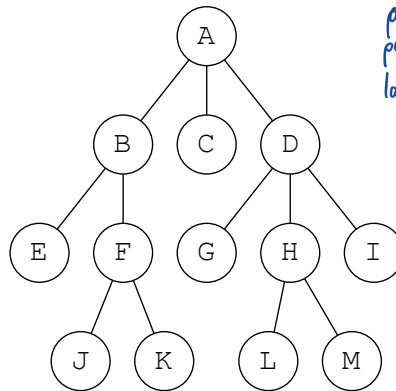
Exercice 5.2 Parcours

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

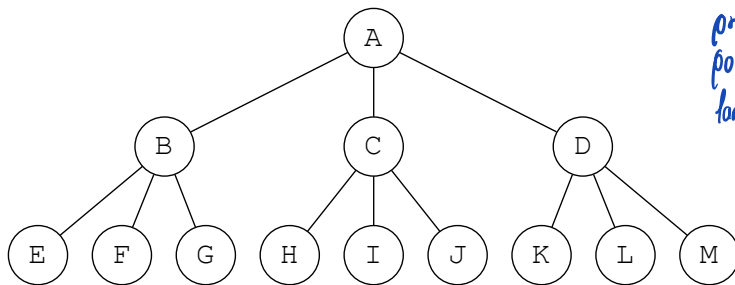
pré: **ABDHIEJCFKLGM** ✓
post: **HIDJEDKLFMGCA** ✓
largeur: **ABCDEFGHIJKLM** ✓



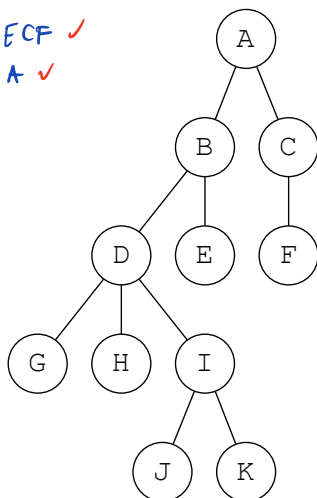
pré: **A B E F J K C D G H L M I** ✓
post: **E J K F D C G L M H I D A** ✓
largeur: **A B C D E G H I K L M** ✓



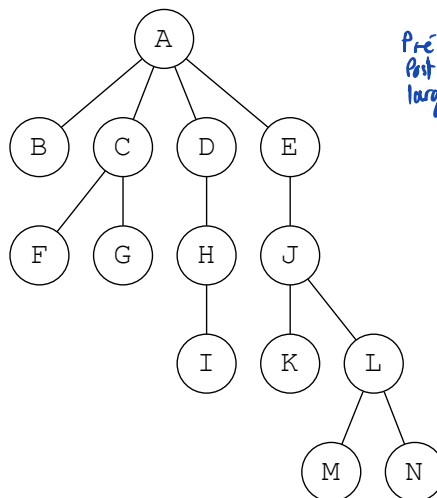
pré: **A B E F G C H I J D K L M** ✓
post: **E F G B H I C K L M D A** ✓
largeur: **A B C D E F G H I J K L M** ✓



pré: **A B D G H I J K E C F** ✓
post: **G H J K I D E B F C A** ✓
largeur: **A B " " J K** ✓

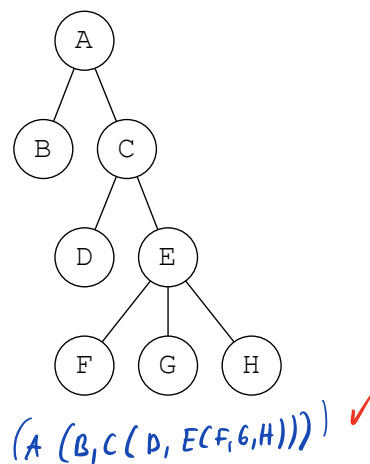
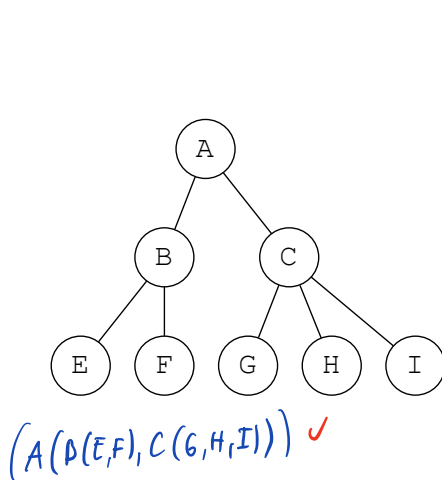
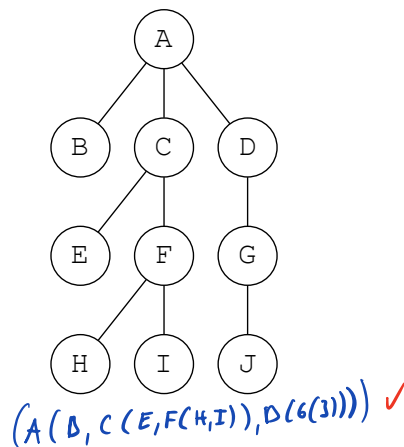
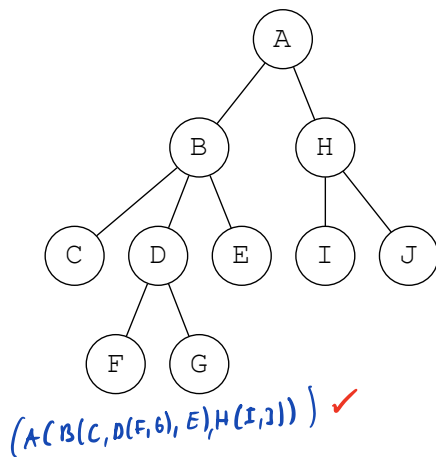


pré: **A B C F G D H I E J K L M N** ✓
post: **B F G C I H D K M N L J E A** ✓
largeur: **A B " " " M N** ✓



Exercice 5.3 Listes imbriquées (1)

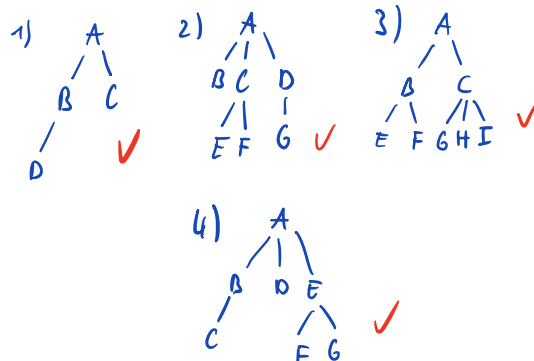
Donnez la représentation sous forme de listes imbriquées des arbres suivants



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
```

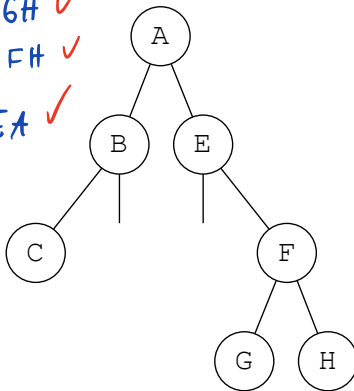
Utilisez comme base le code disponible ici :

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

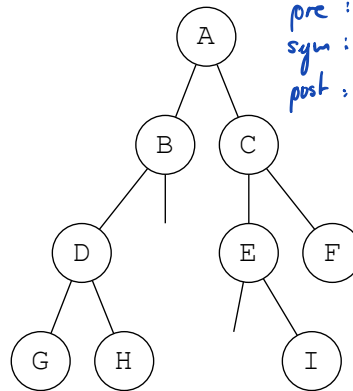
Exercice 5.7 Parcours arbre binaire (1)

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

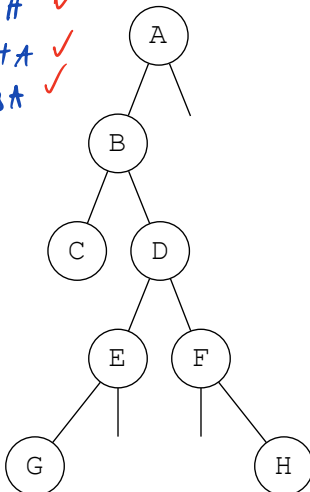
pre : ABCDEFH ✓
sym : CBAEFH ✓
post : CBGFHA ✓



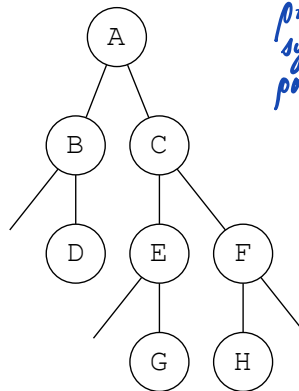
pre : ABDGHEIF ✓
sym : GDHBAEICF ✓
post : GHD BIEFCA ✓



pre : ADCDEGFH ✓
sym : CBGEDFHA ✓
post : CGEHFDBA ✓



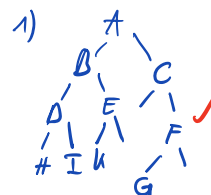
pre : ABDCEGFH ✓
sym : BDAEGCHF ✓
post : ABGEHFC A ✓



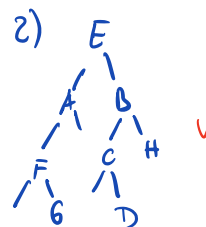
Exercice 5.8 Parcours arbre binaire (2)

Dessinez les arbres binaires dont vous connaissez deux parcours

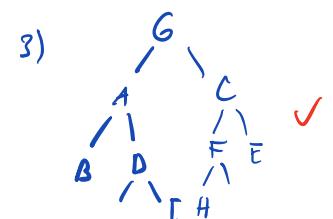
1. pre : A B D H I E K C F G
sym : H D I B K E A C G F



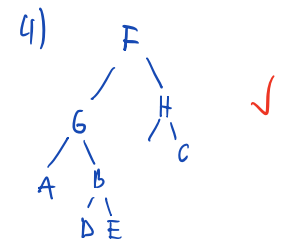
2. sym : F G A E C D B H
post : G F A D C H B E



3. pre : G A B D I C F H E
sym : B A D I G H F C E



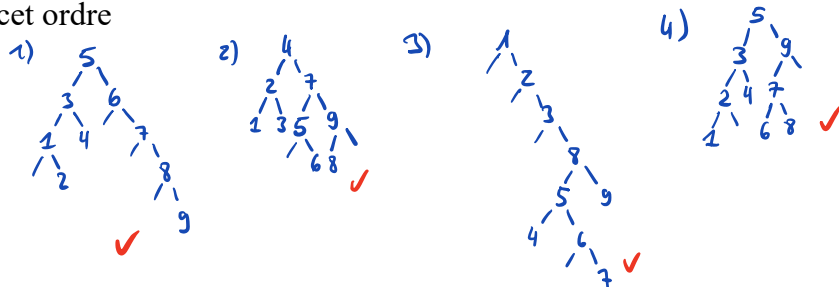
4. sym : A G D B E F H C
post : A D E B G C H F



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

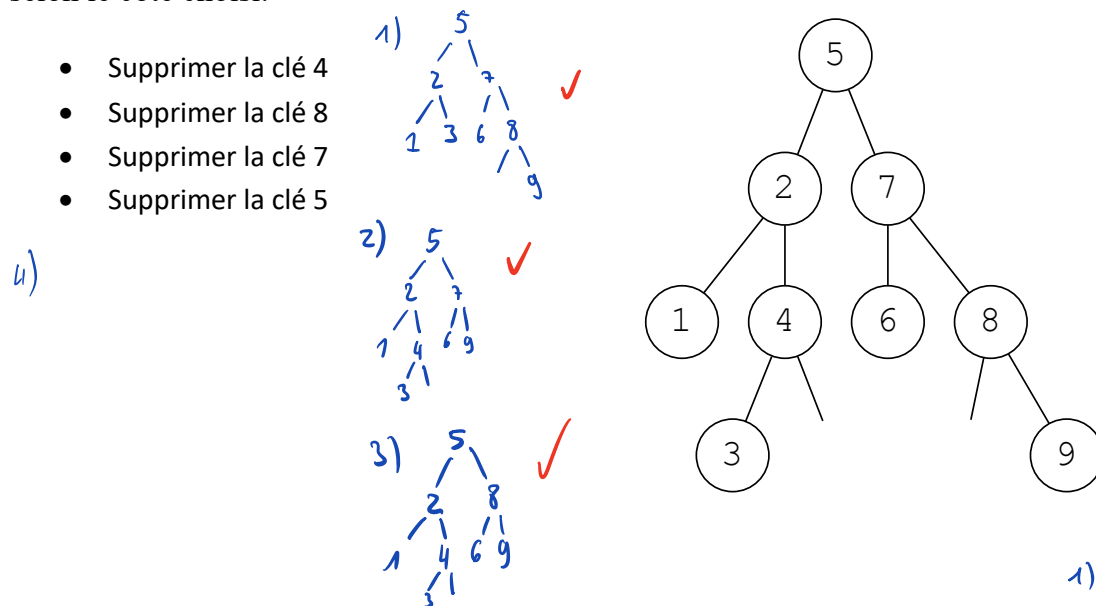
- 5, 3, 1, 6, 7, 4, 8, 2, 9
- 4, 7, 2, 1, 5, 9, 6, 8, 3
- 1, 2, 3, 8, 5, 6, 4, 7, 9
- 5, 3, 9, 7, 6, 8, 2, 4, 1



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.

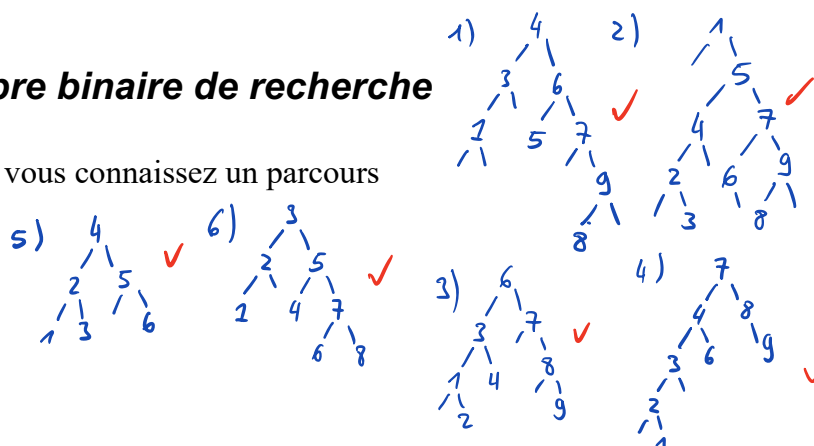
- Supprimer la clé 4
- Supprimer la clé 8
- Supprimer la clé 7
- Supprimer la clé 5



Exercice 5.11 Parcours d'un arbre binaire de recherche

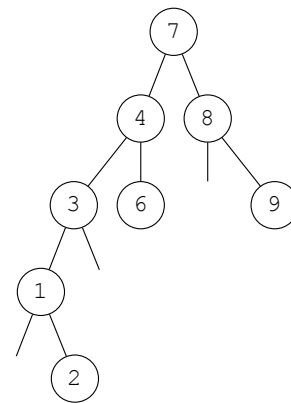
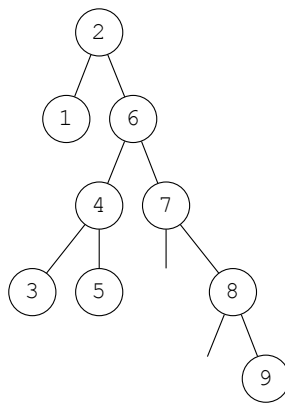
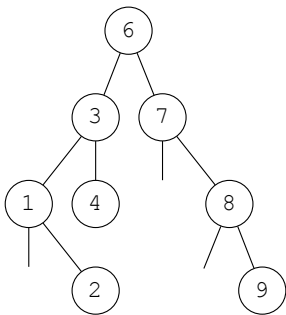
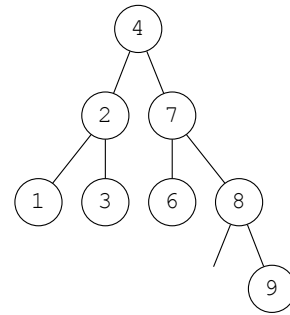
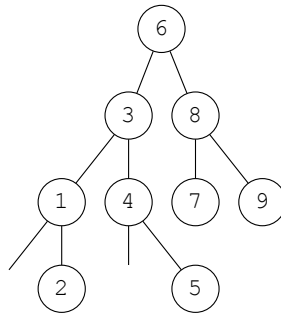
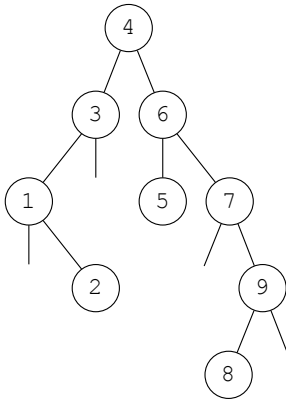
Dessinez les arbres binaires de recherche dont vous connaissez un parcours

- 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

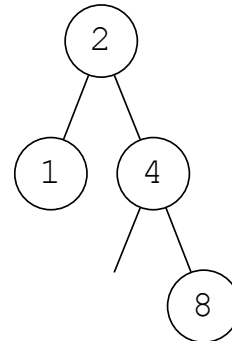
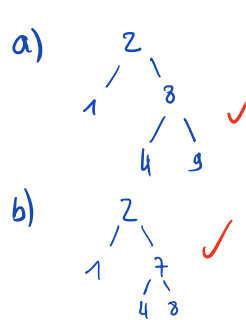
Les arbres suivants sont-ils équilibrés ? Si ce n'est pas le cas, indiquez quels nœuds sont déséquilibrés (sous-arbres de hauteurs différent de plus de 1).



Exercice 5.13 Insertion dans un arbre AVL

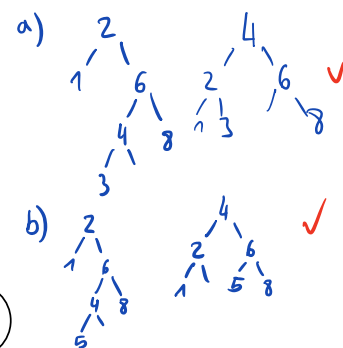
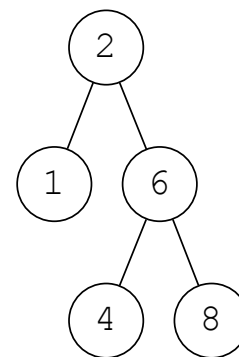
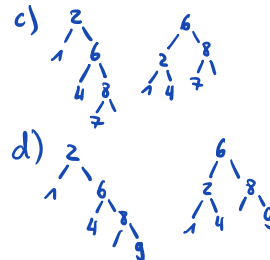
1. En partant à chaque fois de l'arbre AVL ci-contre

- Insérez la clé 9
- Insérez la clé 7



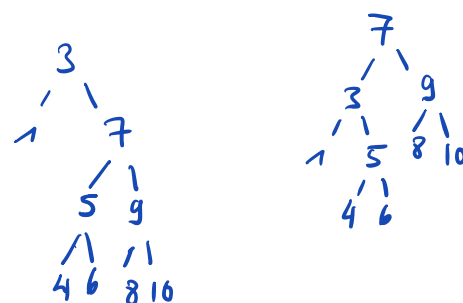
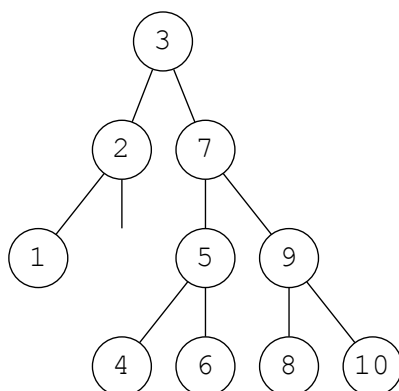
2. En partant à chaque fois de l'arbre AVL ci-contre

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

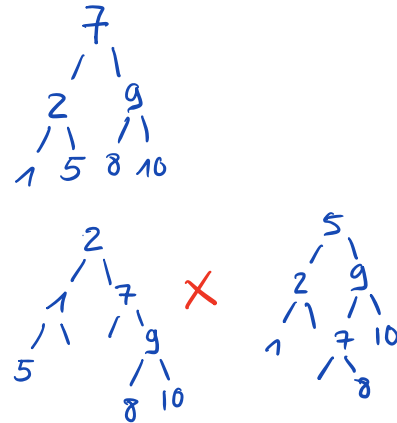
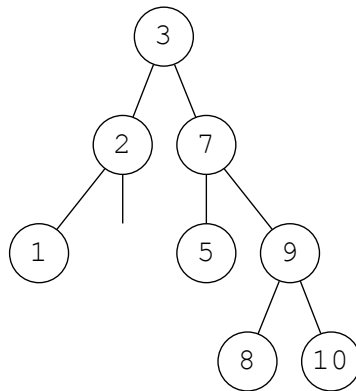


Exercice 5.14 Suppression dans un arbre AVL

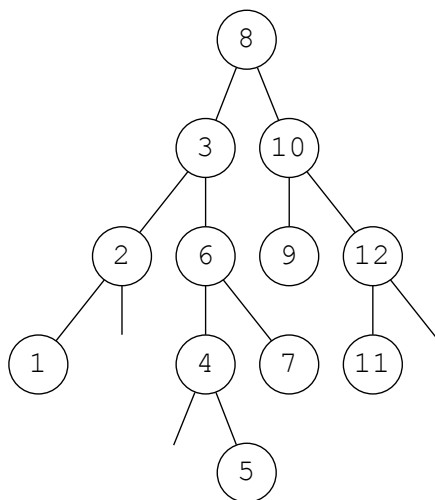
1. Supprimez la clé 2 de l'arbre suivant



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

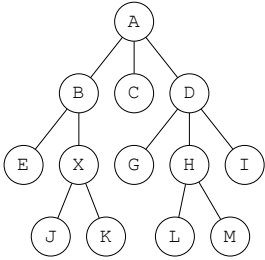
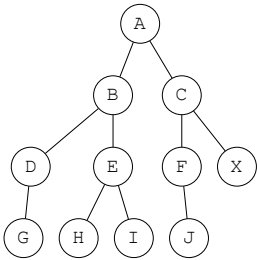
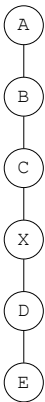
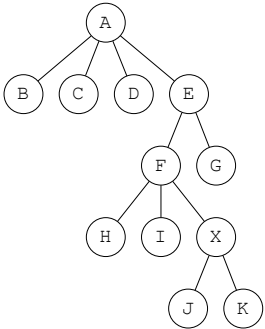


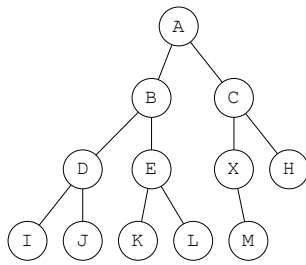
3. Supprimez la clé 9 de l'arbre suivant



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

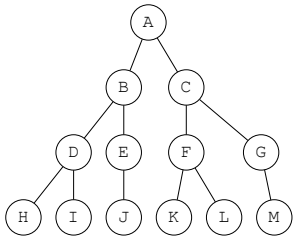
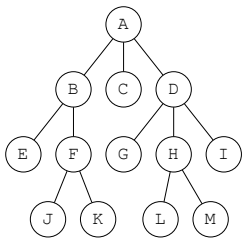
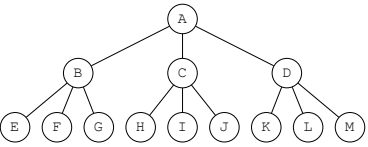
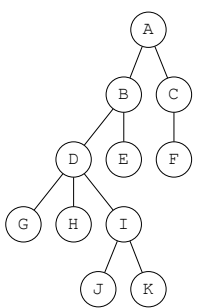
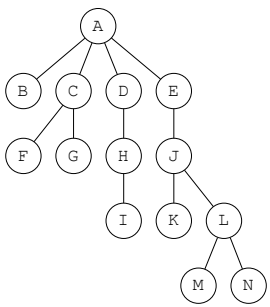
Nœuds internes : A,B,C,D,E,X

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

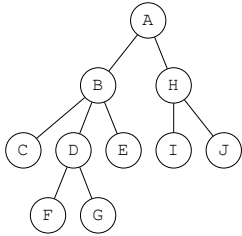
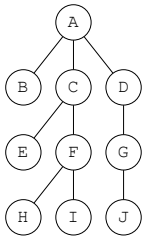
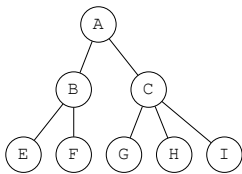
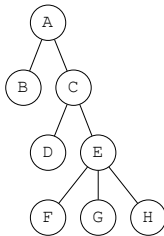
Nœud X de chemin (A,C,X) et de niveau 3

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

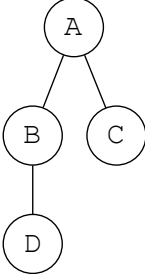
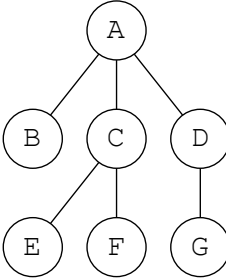
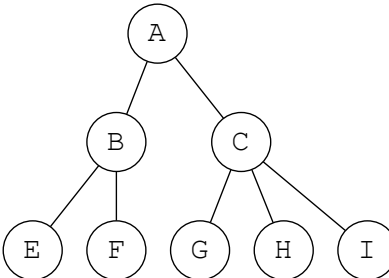
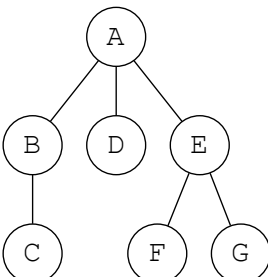
Exercice 5.2 Parcours

 <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)) G --- 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 --- F((F)) C --- G((G)) C --- H((H)) C --- I((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;
}

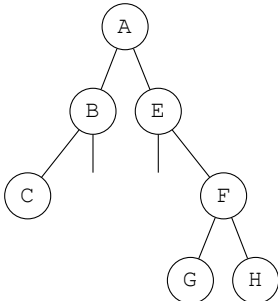
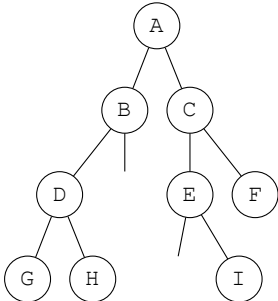
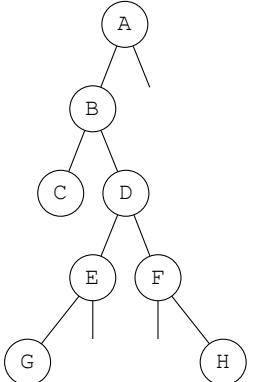
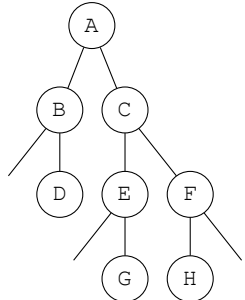
// .first = degre du noeud
// .second = degre de l'arbre
pair<size_t, size_t> degres(const Noeud* n) {
    if (n == nullptr)
        return {0, 0};
    auto dp = degres(n->puîne);
    auto da = degres(n->aîne);
    return {dp.first + 1, max({dp.first + 1, dp.second, da.second})};
}

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)

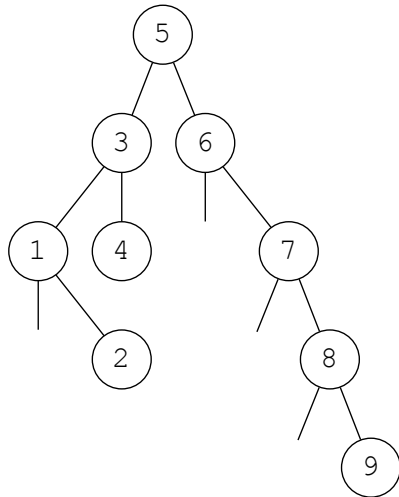
 <pre> graph TD A((A)) --> B((B)) A --> E((E)) B --> C((C)) E --> F((F)) F --> G((G)) F --> H((H)) </pre>	<pre> pre : A B C E F G H sym : C B A E G F H post : C B G H F E A </pre>
 <pre> graph TD A((A)) --> B((B)) A --> C((C)) B --> D((D)) D --> G((G)) D --> H((H)) C --> E((E)) E --> I((I)) E --> F((F))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

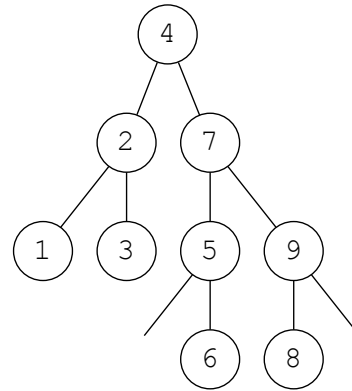
<pre>pre : A B D H I E K C F G sym : H D I B K E A C G F</pre>	<pre>graph TD A((A)) --> B((B)) A --> C((C)) B --> D((D)) B --> E((E)) D --> H((H)) D --> I((I)) E --> K((K)) C --> F((F)) F --> G((G))</pre>
<pre>sym : F G A E C D B H post : G F A D C H B E</pre>	<pre>graph TD E((E)) --> A((A)) E --> B((B))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

Exercice 5.9 Insertions dans un arbre binaire de recherche

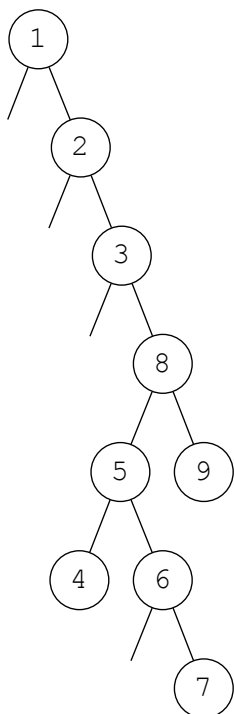
5, 3, 1, 6, 7, 4, 8, 2, 9



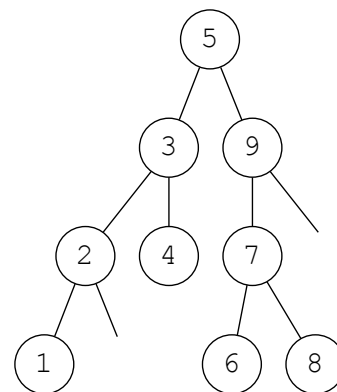
4, 7, 2, 1, 5, 9, 6, 8, 3



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

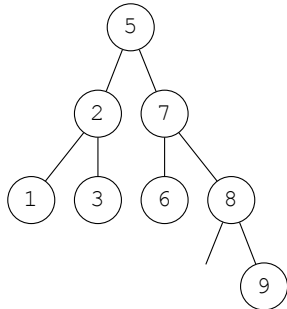


5, 3, 9, 7, 6, 8, 2, 4, 1

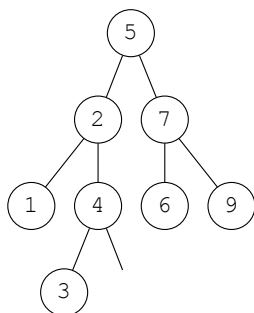


Exercice 5.10 Suppressions d'un arbre binaire de recherche

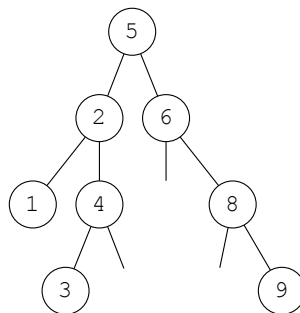
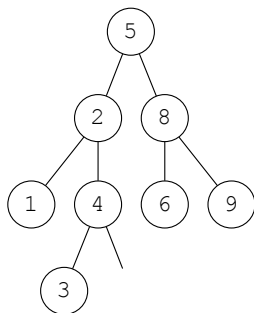
- Supprimer la clé 4



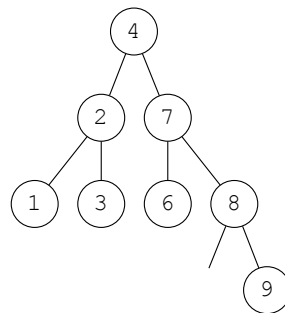
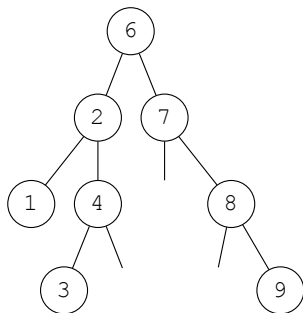
- Supprimer la clé 8



- Supprimer la clé 7 (Hibbard, 2 solutions)



- Supprimer la clé 5 (Hibbard, 2 solutions)

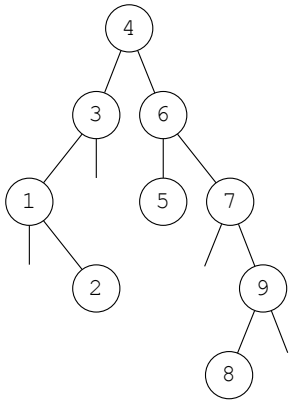
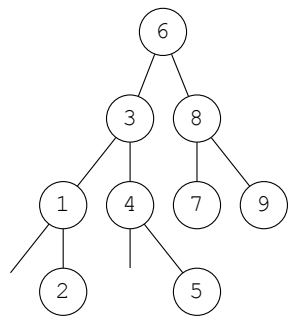
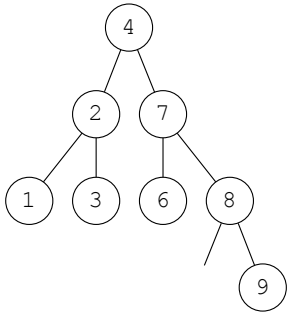


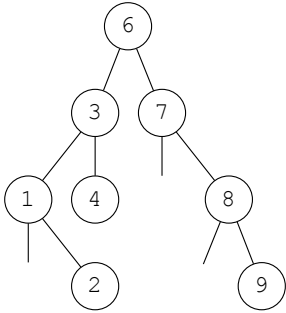
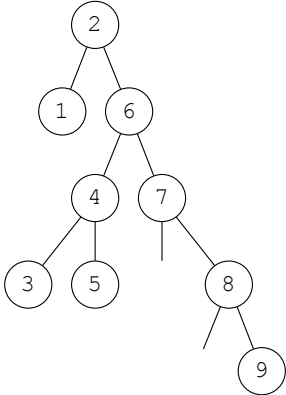
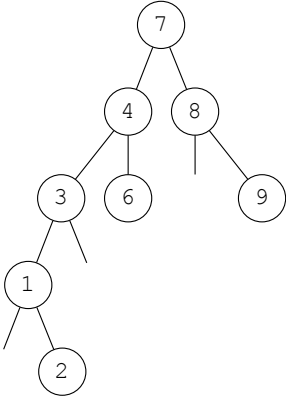
Exercice 5.11 Parcours d'un arbre binaire de recherche

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>

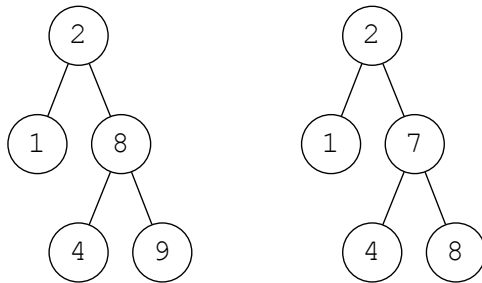
Exercice 5.12 Equilibre

 <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>

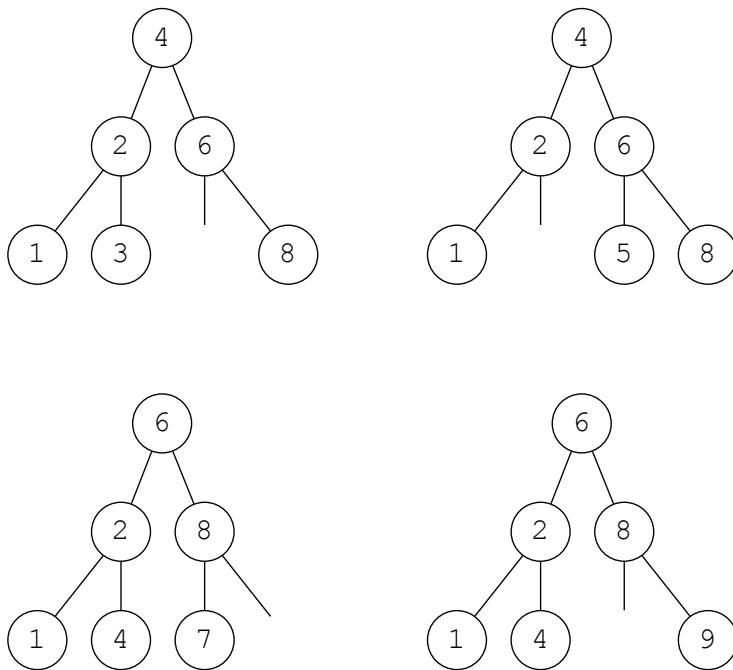
 <pre> graph TD 6((6)) --- 3((3)) 6 --- 7((7)) 3 --- 1((1)) 3 --- 4((4)) 1 --- 2((2)) 7 --- 8((8)) 8 --- 9((9)) </pre>	<p>Le nœud 7 est déséquilibré</p>
 <pre> graph TD 2((2)) --- 1((1)) 2 --- 6((6)) 6 --- 4((4)) 6 --- 7((7)) 4 --- 3((3)) 4 --- 5((5)) 7 --- 8((8)) 8 --- 9((9)) </pre>	<p>Les nœuds 2 et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

