1) notations utilisées

Dans ce chapitre nous allons utiliser les notations suivantes :

Soit un arbre T : T.racine correspond au noeud racine de l'arbre T

Soit un noeud x :

- x.gauche correspond au sous-arbre gauche du noeud x
- x.droit correspond au sous-arbre droit du noeud x
- x.clé correspond à la clé du noeud x

Il faut noter que si le noeud x est une feuille, x.gauche et x.droite sont des arbres vides (NIL)

2) calculer la hauteur d'un arbre

Voici l'algorithme qui permet de calculer la hauteur d'un arbre :

```
VARIABLE
T : arbre
x : noeud

DEBUT
HAUTEUR(T) :
  si T ≠ NIL :
    x ← T.racine
    renvoyer 1 + max(HAUTEUR(x.gauche), HAUTEUR(x.droit))
  sinon :
    renvoyer 0
  fin si
FIN
```

N.B. la fonction max renvoie la plus grande valeur des 2 valeurs passées en paramètre (exemple : max(5,6) renvoie 6

Nous avons ici un algorithme récursif. Vous aurez l'occasion de constater que c'est souvent le cas dans les algorithmes qui travaillent sur des structures de données telles que les arbres.

3) calculer la taille d'un arbre

Nous allons maintenant étudier un algorithme qui permet de calculer le nombre de noeuds présents dans un arbre.

```
VARIABLE
T : arbre
x : noeud

DEBUT
TAILLE(T) :
  si T ≠ NIL :
    x ← T.racine
    renvoyer 1 + TAILLE(x.gauche) + TAILLE(x.droit)
  sinon :
    renvoyer 0
  fin si
FIN
```

4) parcours d'un arbre

a) introduction

Il existe plusieurs façons de parcourir un arbre (parcourir un arbre = passer par tous les noeuds), nous allons en étudier quelques-unes. Le choix du parcours dépend du problème à traiter

b) parcourir un arbre dans l'ordre préfixe

Voici l'algorithme qui va permettre de parcourir un arbre dans l'ordre préfixe :

```
VARIABLE
T : arbre
x : noeud

DEBUT
PARCOURS-PREFIXE(T) :
  si T ≠ NIL :
    x ← T.racine
    affiche x.clé
    PARCOURS-PREFIXE(x.gauche)
    PARCOURS-PREFIXE(x.droit)
  fin si
FIN
```

Comme vous pouvez le constater ci-dessus, dans le cas du parcours préfixe, on affiche chaque noeud avant de parcourir son sous-arbre gauche et son sous-arbre droit.

c) parcourir un arbre dans l'ordre suffixe

Voici l'algorithme qui va permettre de parcourir un arbre dans l'ordre suffixe :

```
VARIABLE
T : arbre
x : noeud

DEBUT
PARCOURS-SUFFIXE(T) :
  si T ≠ NIL :
    x ← T.racine
    PARCOURS-SUFFIXE(x.gauche)
    PARCOURS-SUFFIXE(x.droit)
    affiche x.clé
  fin si
FIN
```

Dans le cas du parcours suffixe, on affiche chaque noeud après avoir parcouru son sous-arbre gauche et son sous-arbre droit.

d) parcourir un arbre dans l'ordre infixe

Voici l'algorithme qui va permettre de parcourir un arbre dans l'ordre infixe :

```
VARIABLE
T : arbre
x : noeud

DEBUT
PARCOURS-INFIXE(T) :
  si T ≠ NIL :
    x ← T.racine
    PARCOURS-INFIXE(x.gauche)
    affiche x.clé
    PARCOURS-INFIXE(x.droit)
  fin si
FIN
```

Dans le cas du parcours infixe, pour un noeud A donné, on parcourra le sous-arbre gauche de A, puis on affichera la clé de A puis enfin, on parcourra le sous-arbre droite de A

e) parcourir un arbre en largeur d'abord

Voici l'algorithme qui va permettre de parcourir un arbre en largeur d'abord :

```
T : arbre
Tg : arbre
Td : arbre
x : noeud
f : file (initialement vide)
```

```
DEBUT
PARCOURS-LARGEUR(T) :
  enfiler(T.racine, f) //on place la racine dans la file
  tant que f non vide :
    x ← defiler(f)
    affiche x.clé
    si x.gauche ≠ NIL :
      Tg ← x.gauche
      enfiler(Tg.racine, f)
    fin si
    si x.droit ≠ NIL :
      Td ← x.droite
      enfiler(Td.racine, f)
    fin si
  fin tant que
FIN
```

Vous noterez aussi que cet algorithme n'utilise pas de fonction récursive. Il est aussi important de bien noter l'utilisation d'une file (FIFO) pour cet algorithme de parcours en largeur.

Dans le cas d'un parcours en largeur d'abord on affiche tous les noeuds situés à une profondeur n avant de commencer à afficher les noeuds situés à une profondeur n+1.

5) algorithmes pour les arbres binaires de recherche

a) Recherche d'une clé dans un arbre binaire de recherche

Nous allons maintenant étudier un algorithme permettant de rechercher une clé de valeur k dans un arbre binaire de recherche. Si k est bien présent dans l'arbre binaire de recherche, l'algorithme renvoie vrai, dans le cas contraire, il renvoie faux.

```
VARIABLE
T : arbre
x : noeud
k : entier
DEBUT
ARBRE-RECHERCHE(T,k) :
  si T == NIL :
    renvoyer faux
  fin si
  x ← T.racine
  si k == x.clé :
    renvoyer vrai
  fin si
```

```
  si k < x.clé :
    renvoyer ARBRE-RECHERCHE(x.gauche,k)
  sinon :
    renvoyer ARBRE-RECHERCHE(x.droit,k)
  fin si
FIN
```

Cet algorithme de recherche d'une clé dans un arbre binaire de recherche ressemble beaucoup à la recherche dichotomique vue en première dans le cas où l'arbre binaire de recherche traité est équilibré. La complexité en temps dans le pire des cas de l'algorithme de recherche d'une clé dans un arbre binaire de recherche équilibré est donc $O(\log_2(n))$. Dans le cas où l'arbre est filiforme, la complexité est O(n). Rappelons qu'un algorithme en $O(\log_2(n))$ est plus "efficace" qu'un algorithme en O(n).

À noter qu'il existe une version dite "itérative" (qui n'est pas récursive) de cet algorithme de recherche :

```
VARIABLE
T : arbre
x : noeud
k : entier
DEBUT
ARBRE-RECHERCHE_ITE(T,k) :
  x ← T.racine
  tant que T ≠ NIL et k ≠ x.clé :
    x ← T.racine
    si k < x.clé :
      T ← x.gauche
    sinon :
      T ← x.droit
    fin si
  fin tant que
  si k == x.clé :
    renvoyer vrai
  sinon :
    renvoyer faux
  fin si
FIN
```

b) Insertion d'une clé dans un arbre binaire de recherche

Il est tout à fait possible d'insérer un noeud y dans un arbre binaire de recherche (non vide) :

```
VARIABLE
T : arbre
x : noeud
y : noeud
```

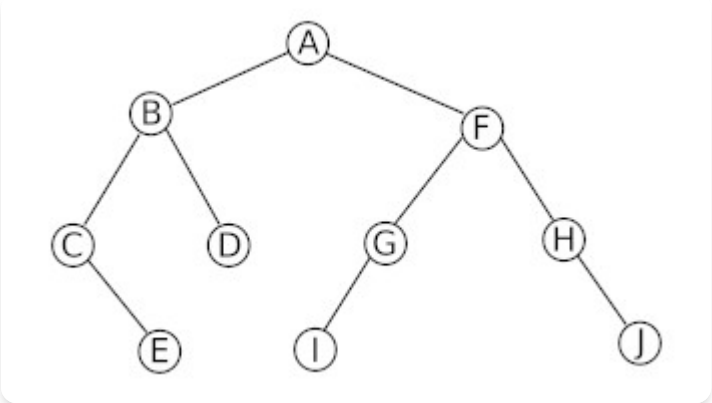
```
DEBUT
ARBRE-INSERTION(T,y) :
  x ← T.racine
  tant que T ≠ NIL :
    x ← T.racine
    si y.clé < x.clé :
      T ← x.gauche
    sinon :
      T ← x.droit
    fin si
  fin tant que
  si y.clé < x.clé :
    insérer y à gauche de x
  sinon :
    insérer y à droite de x
  fin si
FIN
```

c) arbre binaire de recherche et parcours infixe

Il est important de noter qu'un parcours infixe d'un arbre binaire de recherche permet d'obtenir les valeurs des noeuds de l'arbre binaire de recherche dans l'ordre croissant.

activité 8.1

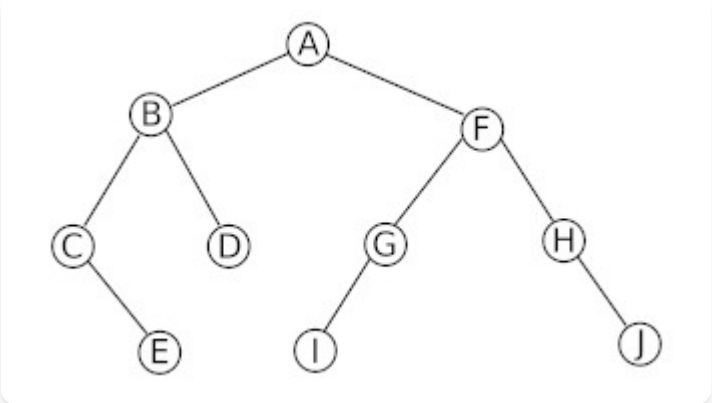
Soit l'arbre suivant :



Appliquez l'algorithme qui permet de calculer le hauteur d'un arbre binaire à l'arbre ci-dessus. Quel résultat obtenez-vous ?

activité 8.2

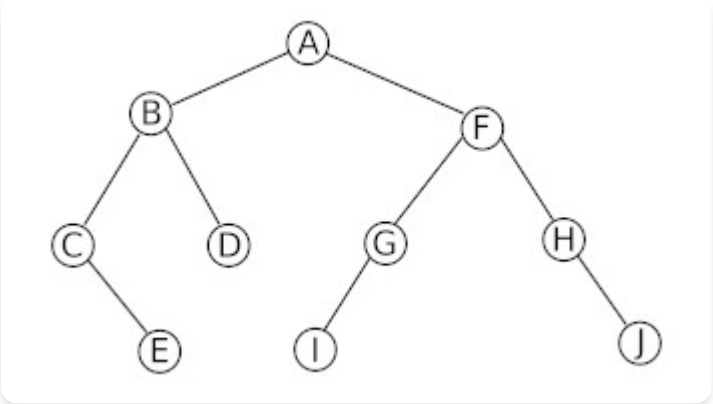
Soit l'arbre suivant :



Appliquez l'algorithme qui permet de calculer la taille d'un arbre binaire à l'arbre ci-dessus. Quel résultat obtenez-vous ?

activité 8.3

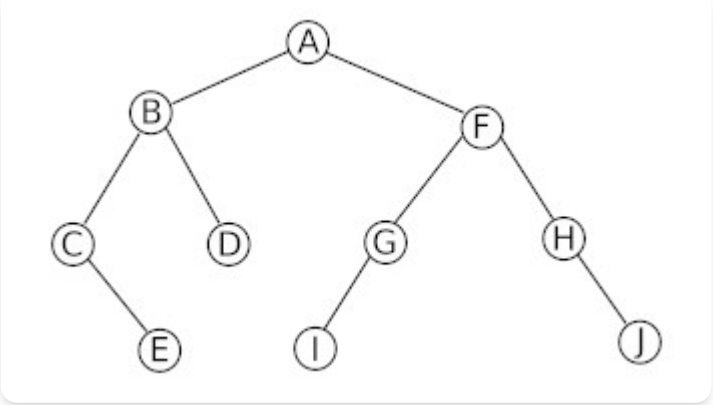
Soit l'arbre suivant :



Appliquez l'algorithme qui permet de trouver un parcours dans l'ordre préfixe à l'arbre ci-dessus. Quel résultat obtenez-vous ?

activité 8.4

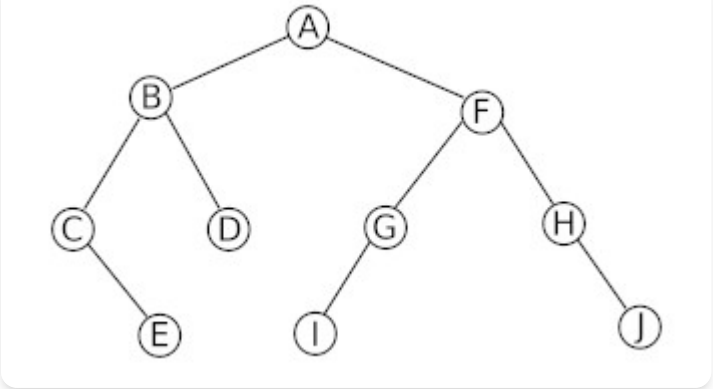
Soit l'arbre suivant :



Appliquez l'algorithme qui permet de trouver un parcours dans l'ordre suffixe à l'arbre ci-dessus. Quel résultat obtenez-vous ?

activité 8.5

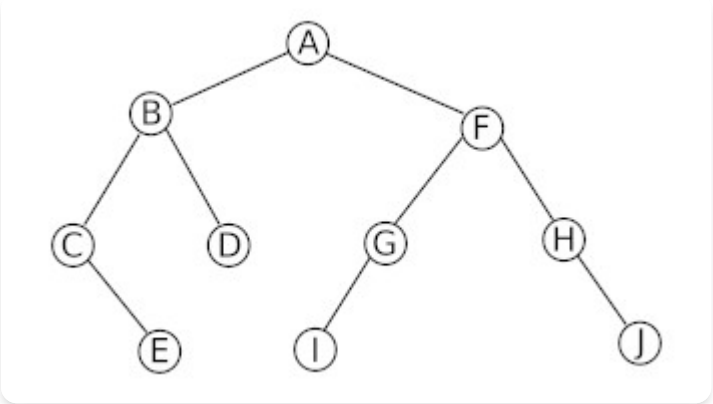
Soit l'arbre suivant :



Appliquez l'algorithme qui permet de trouver un parcours dans l'ordre infixe à l'arbre ci-dessus. Quel résultat obtenez-vous ?

activité 8.6

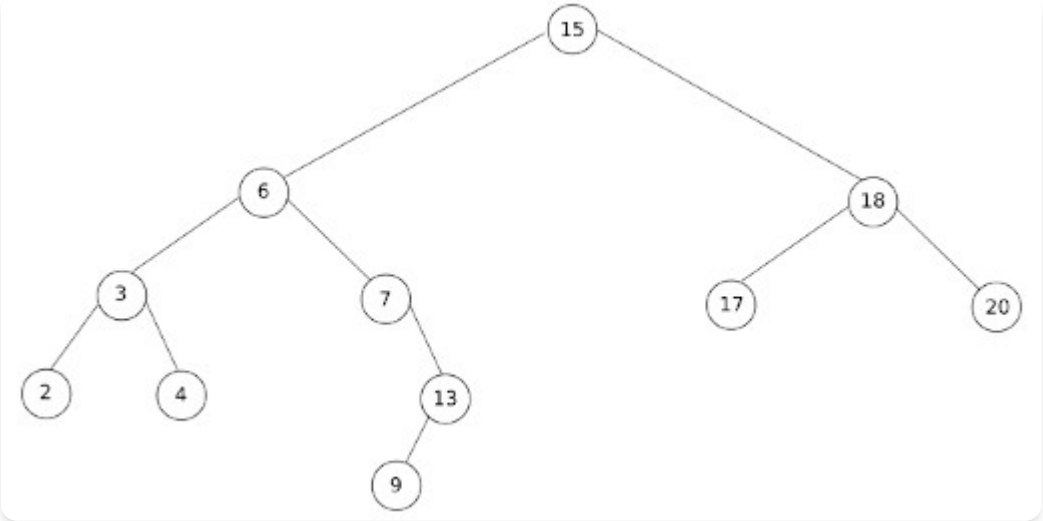
Soit l'arbre suivant :



Appliquez l'algorithme qui permet de trouver un parcours en largeur d'abord à l'arbre ci-dessus. Quel résultat obtenez-vous ?

activité 8.7

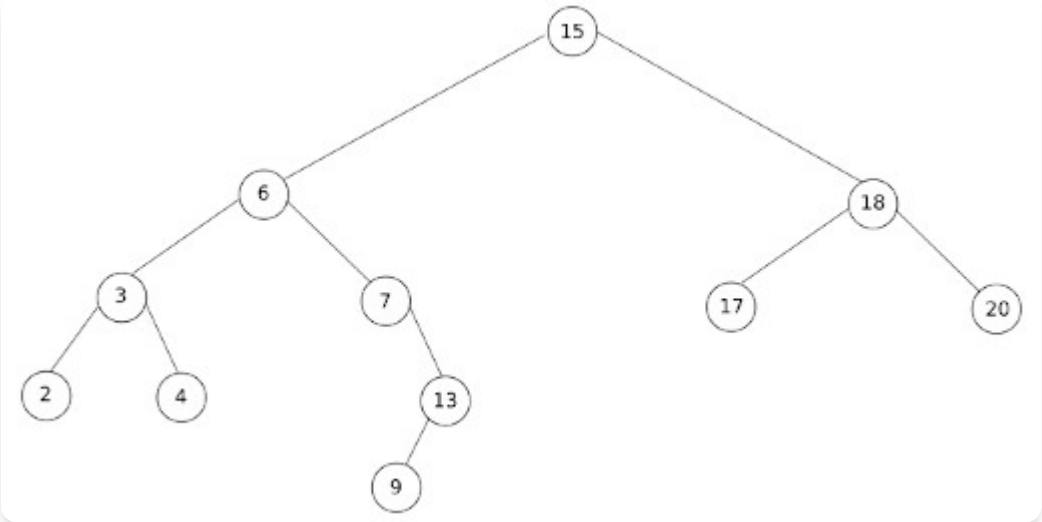
Soit l'arbre suivant :



Appliquez l'algorithme qui permet de trouver un parcours dans l'ordre infixe à l'arbre ci-dessus. Quel résultat obtenez-vous ?

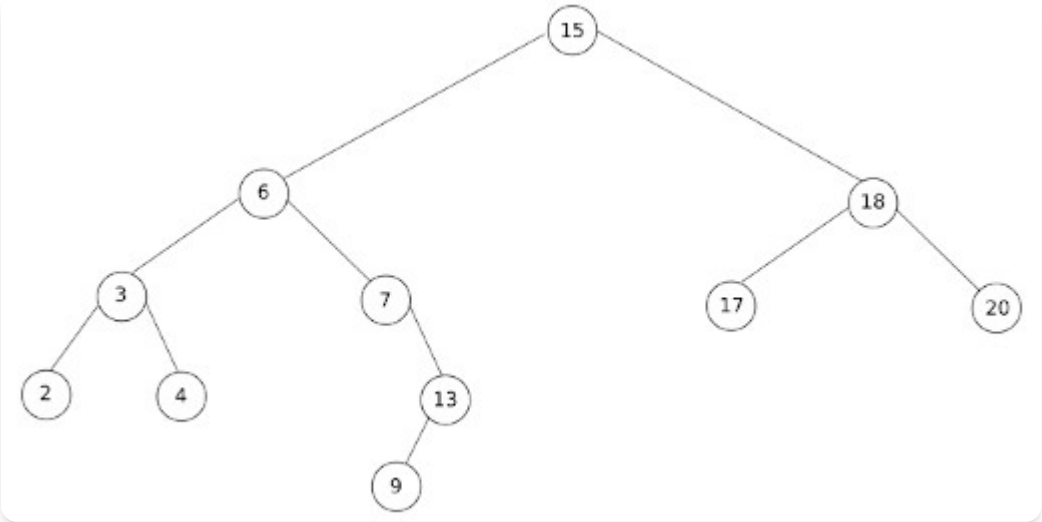
activité 8.8

Appliquez l'algorithme de recherche d'une clé dans un arbre binaire de recherche sur l'arbre ci-dessous. On prendra k = 13. Quel résultat obtenez-vous ?



activité 8.9

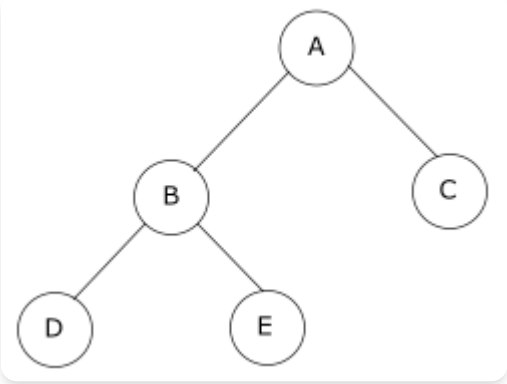
Appliquez l'algorithme de recherche d'une clé dans un arbre binaire de recherche sur l'arbre ci-dessous. On prendra k = 16. Quel résultat obtenez-vous ?



activité 8.10

Dans cette activité, nous allons implémenter des arbres binaires en Python en utilisant des dictionnaires (nous verrons un peu plus tard dans l'année une autre façon de procéder). L'idée est relativement simple : chaque noeud est modélisé à l'aide d'un dictionnaire, ces dictionnaires seront composés de 3 clés (et donc 3 valeurs) : une clé "valeur", une clé "arbre_gauche" et une clé "arbre_droit". La valeur associée à la clé "valeur" sera tout simplement la valeur du noeud. La valeur associé à la clé "arbre_gauche" sera un noeud (donc un autre dictionnaire) si l'arbre gauche existe et None dans le cas contraire. La valeur associée à la clé "arbre_droit" sera un noeud (donc un autre dictionnaire) si l'arbre droit existe et None dans le cas contraire.

L'arbre binaire suivant :



sera implémenté en Python avec le dictionnaire suivant :

```
arbre_1 = {"valeur" : "A", "arbre_gauche" : {"valeur" : "B", "arbre_gauche": {"va
```

que l'on peut aussi représenter comme ceci afin d'améliorer la visibilité :

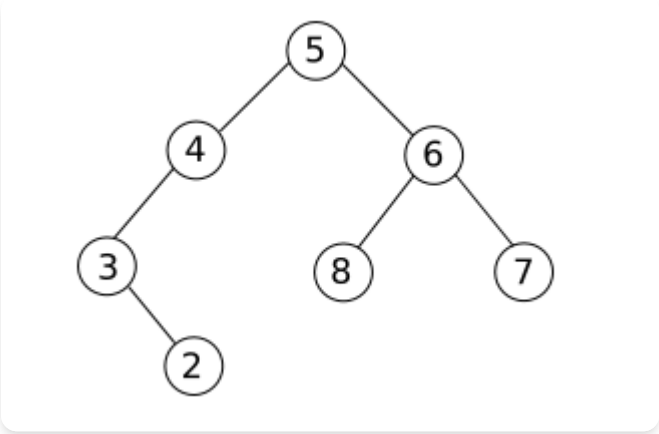
```
arbre_1 = {"valeur":"A",
           "arbre_gauche":
                          {"valeur" : "B",
                          "arbre_gauche":
                                         {"valeur" : "D",
                                         "arbre_gauche": None,
                                         "arbre_droit": None},
                          "arbre_droit": {"valeur" : "E",
                                         "arbre_gauche": None,
                                         "arbre_droit": None}},
           "arbre_droit" : {"valeur" : "C",
                          "arbre_gauche": None,
                          "arbre_droit": None}}
```

1. Écrire en Python une fonction `taille` qui prend en paramètre un arbre `arb` (arbre implémenté sous forme de dictionnaire) et qui renvoie la taille de l'arbre `arb` .
2. Écrire en Python une fonction `hauteur` qui prend en paramètre un arbre `arb` (arbre implémenté sous forme de dictionnaire) et qui renvoie la hauteur de l'arbre `arb` .
3. Écrire en Python une fonction `parcours_prefixe` qui prend en paramètre un arbre `arb` (arbre implémenté sous forme de dictionnaire) et qui affiche les noeuds de l'arbre `arb` dans l'ordre préfixe.
4. Écrire en Python une fonction `parcours_infixe` qui prend en paramètre un arbre `arb` (arbre implémenté sous forme de dictionnaire) et qui affiche les noeuds de l'arbre `arb` dans l'ordre infixe.
5. Écrire en Python une fonction `parcours_suffixe` qui prend en paramètre un arbre `arb` (arbre implémenté sous forme de dictionnaire) et qui affiche les noeuds de l'arbre `arb` dans l'ordre suffixe.

6. Écrire en Python une fonction `parcours_largeur` qui prend en paramètre un arbre `arb` (arbre implémenté sous forme de dictionnaire) et qui affiche les noeuds de l'arbre `arb` en respectant le parcours en largeur.
7. Implémenter en Python un arbre binaire de recherche 'arbre_2' constitué des nombres suivants : 30, 0, 10, 40 et 20
8. Écrire en Python une fonction récursive `arbre_recherche_rec` qui prend en paramètre un arbre binaire de recherche `arb` (arbre implémenté sous forme de dictionnaire) et un entier `k` et qui renvoie `True` si k est bien présent dans l'arbre et False dans le cas contraire.
9. Écrire en Python une fonction non récursive `arbre_recherche_it` qui prend en paramètre un arbre binaire de recherche `arb` (arbre implémenté sous forme de dictionnaire) et un entier `k` et qui renvoie `True` si k est bien présent dans l'arbre et False dans le cas contraire.
10. Écrire en Python une fonction non récursive `insertion` qui prend en paramètre un arbre binaire de recherche `arb` (arbre implémenté sous forme de dictionnaire) et un entier `k` et qui place l'entier `k` dans l'arbre binaire de recherche `arb` .

exercice 8.1

Soit l'arbre binaire A suivant :



1. A propos de l'arbre A :

- Déterminez la profondeur du noeud 6
- Déterminez la hauteur de l'arbre

2. Parcourir l'arbre A dans l'ordre suffixe

3.

- Expliquez pourquoi l'arbre binaire A n'est pas un arbre binaire de recherche
- Modifiez l'arbre binaire A pour qu'il devienne un arbre binaire de recherche (on gardera les mêmes noeuds). On appellera l'arbre binaire obtenu "arbre B"

4. Parcourir l'arbre B dans l'ordre infixe

exercices du bac

- Sujet 1 2021 Exercice 3
- Sujet 2 2021 Exercice 1
- Sujet 6 2021 Exercice 3
- Sujet 8 2021 Exercice 4
- Sujet 1 2022 Exercice 5
- Sujet 4 2022 Exercice 3
- Sujet 5 2022 Exercice 4
- Sujet 9 2022 Exercice 2
- Sujet 10 2022 Exercice 4
- Sujet 11 2022 Exercice 4

- Sujet 13 2022 Exercice 3
- Sujet 14 2022 Exercice 1

Ce qu'il faut savoir

- connaître l'algorithme qui permet de calculer la hauteur d'un arbre (voir cours)
- connaître l'algorithme qui permet de calculer la taille d'un arbre (voir cours)
- connaître les algorithmes qui permettent de parcourir un arbre : ordre infixe, ordre préfixe, ordre suffixe, en largeur d'abord (voir cours)
- connaître l'algorithme qui permet de rechercher une clé dans un arbre binaire de recherche (voir cours), savoir que cet algorithme à une complexité en $O(\log_2(n))$ dans le cas d'un arbre binaire de recherche équilibré et $O(n)$ dans le cas d'un arbre binaire de recherche filiforme.
- connaître l'algorithme qui permet d'insérer une clé dans un arbre binaire de recherche (voir cours)

Ce qu'il faut savoir faire

Vous devez être capable d'implémenter tous ces algorithmes en Python (voir activité)