ÉVALUATION SUR LES ARBRES ET ARBRES BINAIRES

Partie 1 - Vocabulaire sur les arbres

Voici un arbre enraciné :

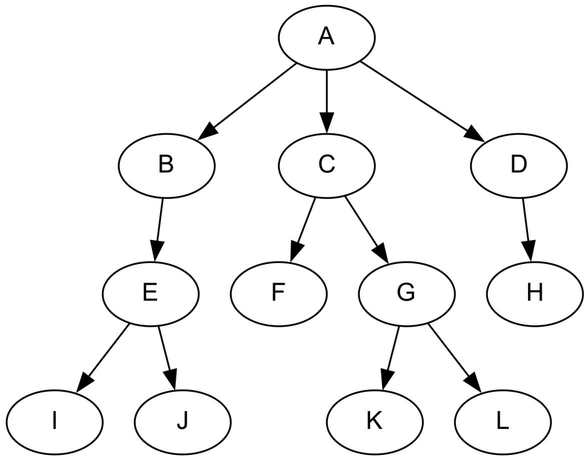


Figure 1: Un arbre enraciné

On considèrera que la **profondeur** du **nœud racine** est de **0**.
Chaque **nœud** possède une **étiquette**, qui est une *lettre* entre *A* et *L*.

Exercice 1

Répondre aux questions suivantes :

1. Quelle est le **taille** de cet arbre ?
2. Combien y a t-il d'**arêtes** dans cet arbre ?
3. Quel est le **nœud racine** de l'arbre (indiquer l'étiquette associée à ce nœud) ?
4. Quelle est la **profondeur** du nœud **F** ?
5. Quelles sont les **feuilles** de cet arbre ?
6. Quelle est la **hauteur** de cet arbre ?

Correction exercice 1

1. La **taille** correspond au **nombre de nœuds** dans l'arbre, ici **12**.
2. Le **nombre d'arêtes** correspond à la **taille** de l'arbre moins **1**, car chaque nœud est relié à un nœud père à l'exception de la racine de l'arbre. Ici, on compte donc **11** arêtes.
3. Le **nœud racine** de l'arbre est le nœud d'*étiquette* **A**.
4. La **profondeur** du nœud **F** est de **2**.
5. Une **feuille** est un **nœud qui n'a pas de nœuds fils**. Ici, on a donc les feuilles **I**, **J**, **F**, **K**, **L** et **H**.
6. La **hauteur** d'un arbre est la **profondeur des feuilles les plus profondes** de l'arbre (ici les feuilles **I**, **J**, **K** et **L**). La **hauteur** est donc de **3**.

Partie 2 - Dessiner un arbre binaire

Voici un **tableau** contenant les données d'un **arbre binaire** :

Nœud	Étiquette du nœud	Racine du sous-arbre gauche	Racine du sous-arbre droit
A	*	B	C
B	8		
C	/	D	E
D	+	F	G
E	3		
F	6		
G	6		

Exercice 2

Représenter l'**arbre binaire** correspondant.
La racine de l'arbre est le **nœud A** ayant pour étiquette $*$.
Votre arbre doit donc commencer de la manière suivante :

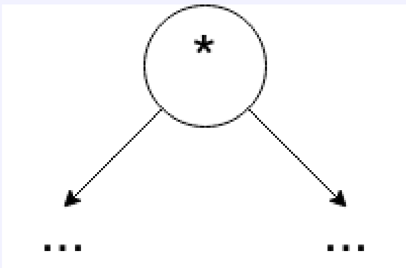


Figure 2: Arbre d'une expression arithmétique

Correction exercice 2

Voici l'arbre obtenu :

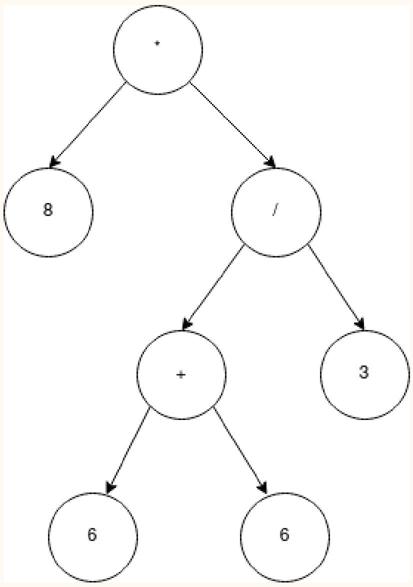


Figure 3: Arbre d'une expression arithmétique

Exercice 3

1. Quelle est la **taille** de l'arbre binaire ainsi dessiné ?
2. Quelle est la **hauteur** de l'arbre, en considérant que la racine est de **profondeur 1** ?
3. Dans un arbre binaire, l'**ordre des sous-arbres** a-t-il une importance ?
   Pour quels opérateurs inverser le *sous-arbre gauche* et *droit* poserait un problème ?
4. Quel est le résultat de l'opération représentée par cet arbre ?

Correction exercice 3

1. La **taille** de cette arbre est de **7** (car on compte **7** nœuds dans l'arbre).
2. Cet arbre a une **hauteur** de **4** (en considérant que la racine est de **profondeur 1**).
3. Oui, inverser le **sous-arbre gauche** et le **sous-arbre droit** donne un arbre binaire **différent**, dont les données n'ont **pas le même sens**. Dans le cas d'un arbre d'une expression arithmétique, si l'on inverse les sous-arbres pour les opérateurs **-** (soustraction) et / (division), le calcul n'est alors plus le même.

Partie 3 - Parcours d'arbres binaires

Voici un **arbre binaire** :

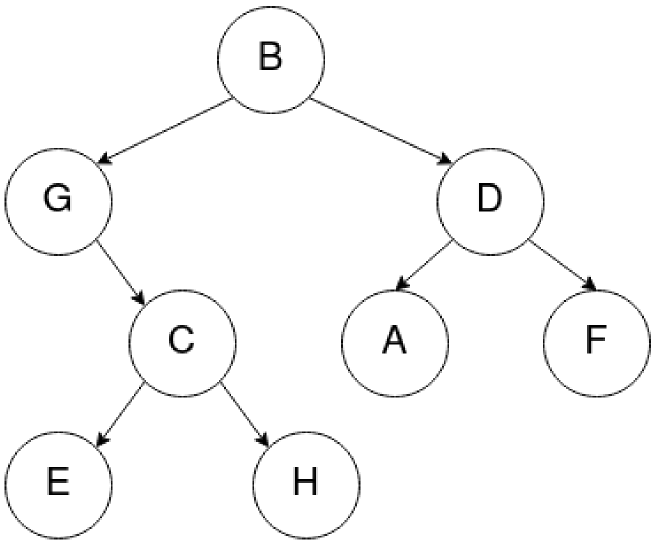


Figure 4: Un arbre binaire

Exercice 4

Indiquer l'**ordre de visite** des **nœuds** lors d'un **parcours en largeur**.

Correction exercice 4

On visite les **nœuds niveau par niveau** et **de gauche à droite**.
On obtient donc l'ordre de visite suivant : **[B, G, D, C, A, F, E, H]**.

Voici 3 algorithmes de **parcours en profondeur** :

Parcours A

Précondition : L'arbre n'est pas vide
1. On `visite` le **nœud racine** de l'arbre.
2. On effectue le `parcours A` du **sous-arbre gauche** s'il est NON vide.
3. On effectue le `parcours A` du **sous-arbre droit** s'il est NON vide.

Parcours B

Précondition : L'arbre n'est pas vide
1. On effectue le `parcours B` du **sous-arbre gauche** s'il est NON vide.
2. On effectue le `parcours B` du **sous-arbre droit** s'il est NON vide.
3. On `visite` le **nœud racine** de l'arbre.

Parcours C

Précondition : L'arbre n'est pas vide
1. On effectue le `parcours C` du **sous-arbre gauche** s'il est NON vide.
2. On `visite` le **nœud racine** de l'arbre.
3. On effectue le `parcours C` du **sous-arbre droit** s'il est NON vide.

Exercice 5

Quel parcours (**A**, **B** ou **C**) correspond à un **ordre préfixe**, à un **ordre infixe** et à un **ordre suffixe** ?

Correction exercice 5

Parcours A : ordre **préfixe** (*Racine-Gauche-Droite*).
Parcours B : ordre **suffixe** (*Gauche-Droite-Racine*).
Parcours C : ordre **infixe** (*Gauche-Racine-Droite*).

Exercice 6

En reprenant l'**arbre binaire** de la *figure 3* :
1. Indiquer l'**ordre de visite** des **nœuds** lors d'un **parcours préfixe**.
2. Indiquer l'**ordre de visite** des **nœuds** lors d'un **parcours suffixe**.
3. Indiquer l'**ordre de visite** des **nœuds** lors d'un **parcours infixe**.

Correction exercice 6

1. Parcours **préfixe** : **[B, G, C, E, H, D, A, F]**
2. Parcours **suffixe** : **[E, H, C, G, A, F, D, B]**
3. Parcours **infixe** : **[G, E, C, H, B, A, D, F]**

Partie 4 - Implémentation d'un arbre binaire

On propose une classe `Arbre` représentant un **arbre binaire** et définie comme suit :

```
class Arbre:
  def __init__(self, valeur=None, gauche=None, droite=None):
    self.v = valeur
    self.g = gauche
    self.d = droite
```

On définit, en dehors de la classe, deux **fonctions d'interface** `nvABV` et `nvAB` qui renvoient respectivement un **nouvel arbre binaire vide** et un **nouvel arbre binaire non vide** :

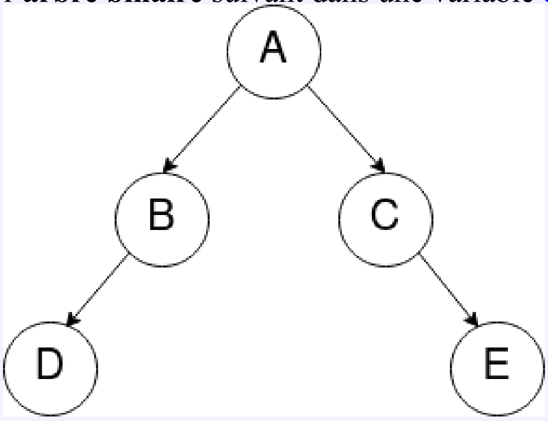
```
def nvABV() -> Arbre:
  ''' Renvoie un nouvel arbre binaire vide. '''
  return Arbre()

def nvAB(valeur: str, gauche: Arbre, droite: Arbre) -> Arbre:
  ''' Renvoie un nouvel arbre binaire non vide, caractérisé par
```

```
    l'étiquette de son noeud racine, un ss-arbre gauche et ss-arbre droit.
    '''
    return Arbre(valeur, gauche, droite)
```

Exercice 7

En réutilisant uniquement **les deux fonctions ci-dessus**, indiquez la (ou les) instruction(s) à saisir pour créer l'**arbre binaire** suivant dans une variable `ab` :



Note : Vous pouvez utiliser des variables supplémentaires si besoin.

```
ab = nvAB( #À COMPLETER# )
```

Correction exercice 7

On peut créer cet arbre **en une instruction** de la manière suivante :

```
ab = nvAB('A',
          nvAB('B',
               nvAB('D', nvABV(), nvABV()),
               nvABV()),
          nvAB('C',
               nvABV(),
               nvAB('E', nvABV(), nvABV())))
```

On peut également décomposer les sous-arbres en **plusieurs variables** pour simplifier la création :

```
# Création des feuilles
D = nvAB('D', nvABV(), nvABV())
E = nvAB('E', nvABV(), nvABV())
# Création des sous-arbres de racines B et C
B = nvAB('B', D, nvABV())
C = nvAB('C', nvABV(), E)
# Création de l'arbre final
ab = nvAB('A', B, C)
```

Exercice 8

Implémentez les **fonctions** suivantes :

- `gauche(ab: Arbre)` et `droite(ab: Arbre)` qui renvoient respectivement le **sous-arbre gauche** et le **sous-arbre droit** d'un **arbre binaire** donné.
- `est_vide(ab: Arbre)-> bool` : Renvoie *True* si l'**arbre binaire** donné est vide, *False* sinon.
- `taille(ab: Arbre)-> int` : Renvoie la **taille** d'un **arbre binaire** donné.
- `hauteur(ab: Arbre)-> int` : Renvoie la **hauteur** d'un **arbre binaire** donné, la hauteur de l'arbre vide étant de **-1**.

Correction eercice 8

```python
def gauche(ab: Arbre) -> Arbre:
  return ab.g

def droite(ab: Arbre) -> Arbre:
  return ab.d

def est_vide(ab: Arbre) -> bool:
  return ab.v == None
  # On aurait pu écrire:
  # return ab.v == None and ab.g == None and ab.d == None,
  # mais cela n'est pas nécessaire car on sait que si la valeur de la racine ab.v vaut None,
  # les deux autres attributs valent également None et il s'agit bien d'un arbre binaire vide.

def taille(ab: Arbre) -> int:
  if est_vide(ab):
    return 0
  else:
    return 1 + taille(gauche(ab)) + taille(droite(ab))

  # A la place de gauche(ab), on peut aussi écrire ab.g
  # A la place de droite(ab), on peut aussi écrire ab.d
  # Les fonctions gauche et droite que l'on a créé permettent d'encapsuler les données et de ne pas
  # faire directement appel aux attributs de l'arbre binaire.

def hauteur(ab: Arbre) -> int:
  if est_vide(ab):
    return -1
  else:
    return 1 + max(hauteur(gauche(ab)), hauteur(droite(ab)))
```