ÉVALUATION SUR LES ARBRES ET ARBRES BINAIRES

Partie 1 - Vocabulaire sur les arbres

Voici un arbre enraciné :

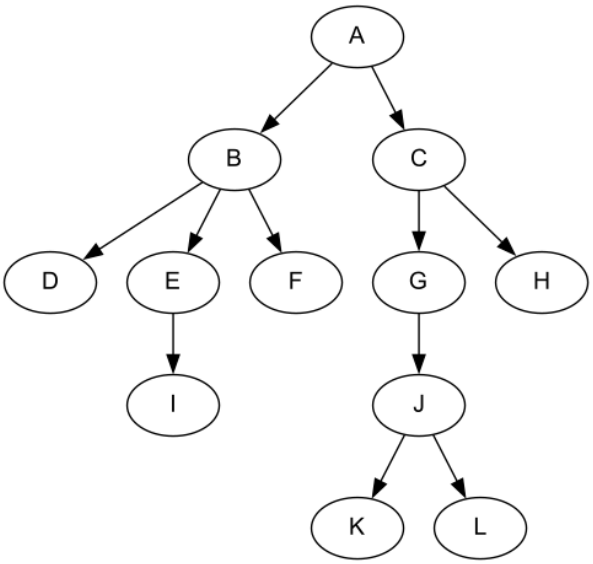


Figure 1: Un arbre enraciné

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 **I** ?
5. Quelles sont les **feuilles** de cet arbre ?
6. Quelle est la **hauteur** de cet arbre ?
7. Proposez une relation calculant le **nombre d'arêtes** (qu'on notera na) d'un arbre en fonction de sa **taille** (qu'on notera n).

Partie 2 - Dessiner un arbre binaire

On a représenté ci-dessous les cinq **arbres binaires** possibles pour une **taille** $n = 3$:

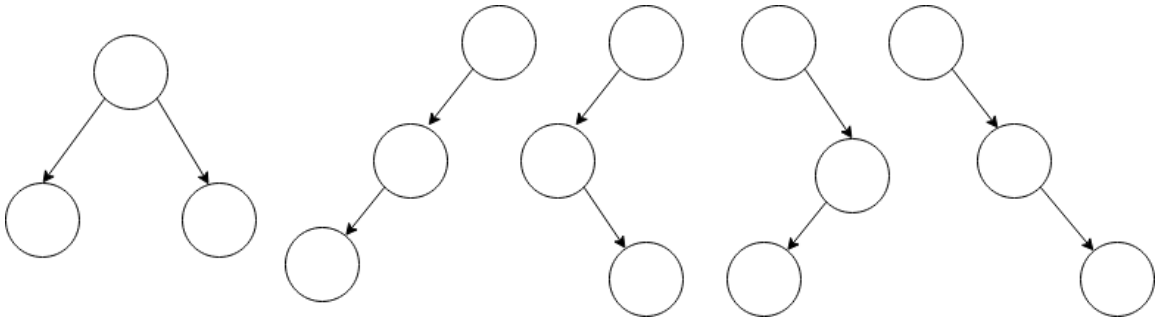


Figure 2: Arbres binaires de taille 3

Exercice 2

Pour les questions suivantes, on considèrera que le **nœud racine** d'un arbre est de **profondeur 1**.

1. Quelle est donc la **hauteur minimale** et **maximale** possible pour un **arbre binaire** de **taille** $n = 3$?
2. À votre tour, dessinez **6 arbres binaires différents** de **taille** $n = 4$.
3. Pour une **taille** $n = 5$, dessinez :
   - un **arbre binaire** ayant **la plus petite hauteur possible**,
   - un **arbre binaire** ayant **la plus grande hauteur possible**.
3. Déduire de la question précédente un encadrement de la **hauteur** (notée h) pour un arbre de **taille** $n = 5$.

Voici un **arbre binaire** représentant une **opération arithmétique** :

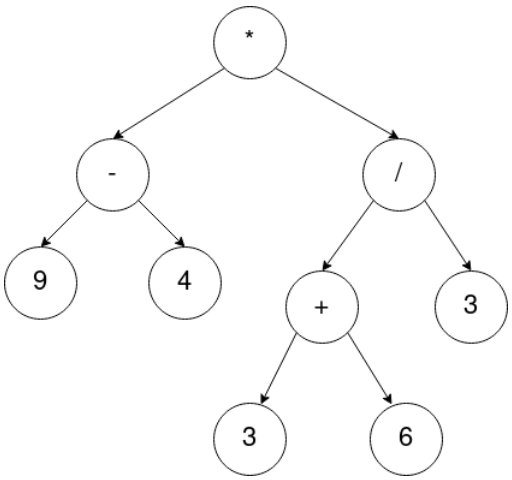


Figure 3: Arbre d'une expression arithmétique

Exercice 3

1. Que pouvez-vous dire des **feuilles** de cet arbre ?
2. Quel est le résultat de l'opération représentée par cet arbre ?
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 ? Justifier.

Partie 3 - Parcours d'arbres binaires

Voici un **arbre binaire** :

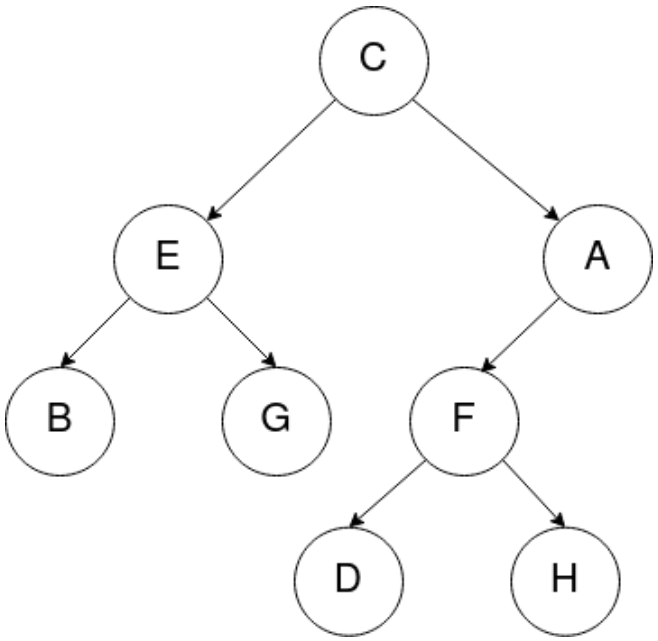


Figure 4: Un arbre binaire

Exercice 4

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

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

Parcours A

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

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 effectue le `parcours C` du **sous-arbre droit** s'il est NON vide.
3. On `visite` le **nœud racine** de l'arbre.

Exercice 5

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

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**.

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_racine=None, gauche=None, droite=None):
    self.v = valeur_racine
    self.g = gauche
    self.d = droite
```

Le **constructeur** de la classe **Arbre** peut être appelé **uniquement des deux manières suivantes** :

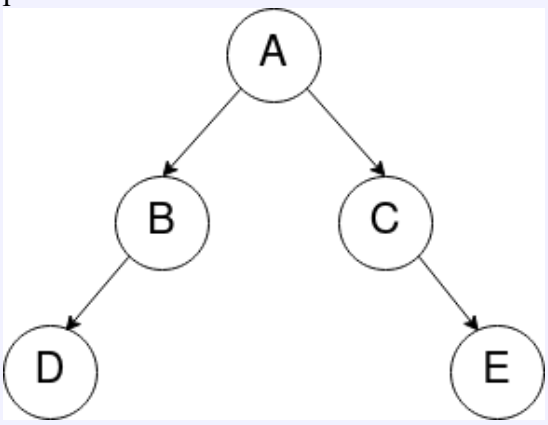
- `Arbre()` : Crée un **nouvel arbre binaire vide**. Les attributs de l'objet auront la valeur `None`.
- `Arbre(valeur_racine, gauche, droite)` : Crée un **nouvel arbre binaire non vide** caractérisé par la **valeur** (= *clé* ou *étiquette*) de sa **racine** (une *chaîne de caractères*), ainsi qu'un **sous-arbre gauche** et **droit** (de type *Arbre*).

Exercice 7

Écrivez **deux fonctions d'interface** `nvABV()` et `nvAB(valeur_racine: str, gauche: Arbre, droite: Arbre)`, qui renvoient respectivement un **nouvel arbre binaire vide** et un **nouvel arbre binaire non vide** (vous devez donc utiliser la classe `Arbre`).

Exercice 8

En réutilisant **uniquement les deux fonctions** `nvABV` et `nvAB`, 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.
Rappel : Un **arbre binaire** a **toujours deux sous-arbres** (pouvant être **vides**). Pour créer une **feuille** contenant la valeur **V**, on écrira donc `nvAB('V', nvABV(), nvABV())`.

```
ab = nvAB( #À COMPLETER# )
```

Exercice 9

Implémentez les **fonctions** suivantes :

- `valeur_racine(ab: Arbre)`, `gauche(ab: Arbre)` et `droite(ab: Arbre)` qui renvoient respectivement la **valeur de la racine**, 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.