Exercices arbre binaire correction

Christophe Viroulaud

Terminale - NSI

Algo 08

Exercices arbre binaire correction

Exercice 1

Exercice 2

Exercice 3

Sommaire

1. Exercice 1

2. Exercice 2

3. Exercice 3

Exercice 1

1.
   - préfixe : $\times$ - 12 8 + 7 9
   - infixe : 12 - 8 $\times$ 7 + 9
   - postfixe : 12 8 - 7 9 + $\times$
2. 64
3. Parcours infixe

Sommaire

1. Exercice 1
2. Exercice 2
3. Exercice 3

Exercice 2

1.
   - en largeur : 1 2 3 4 5 6 7 8 9 10 11 12 13
   - préfixe : 1 2 4 8 5 3 6 9 10 12 13 7 11
   - infixe : 4 8 2 5 1 9 6 12 10 13 3 11 7
   - postfixe : 8 4 5 2 9 12 13 10 6 11 7 3 1
2. La hauteur est 4.
3. Cet arbre est équilibré car la hauteur de chaque sous-arbre gauche diffère au plus de 1 de chaque sous-arbre droit.
4. Cet arbre n'est pas complet car tous les niveaux ne sont pas remplis.

Sommaire

1. Exercice 1

2. Exercice 2

3. Exercice 3

Exercice 3


```
1 class Arbre_binaire:
2     def __init__(self, h: int):
3         self.hauteur = h
4         self.arbre = [None for i in range(2**(h+1)-1)]
5         self.arbre[0] = "r"
```

Code 1 – Constructeur

```python
def get_taille(self) -> int:
    taille = 0
    for elt in self.arbre:
        # on évite les noeuds vides
        if elt is not None:
            taille += 1
    return taille
```

Remarque

Pour alléger le diaporama les *docstring* ne sont pas présentes. Elles sont écrites dans le fichier Python.

```python
def get_indice(self, chaine: str) -> int:
    i = 0
    while self.arbre[i] != chaine:
        i = i+1
    return i
```

Remarque

Tous les nœuds sont distincts.

```
1 def inserer(self, pere: str, gauche: str, droit: str) -> None:
2     i_pere = self.get_indice(pere)
3     #assert 2*i_pere+2 < len(self.arbre)
4     assert 2*i_pere+2 < 2**(self.hauteur+1)-1, "dépassement de
      taille"
5     self.arbre[2*i_pere+1] = gauche
6     self.arbre[2*i_pere+2] = droit
```

Remarque

La taille du tableau est fixée.

```
1 def prefixe(self, position: int, parcours: list) -> None:
2     if position < len(self.arbre) and \
3                     self.arbre[position] is not None:
4         parcours.append(self.arbre[position])
5         self.prefixe(2*position+1, parcours)
6         self.prefixe(2*position+2, parcours)
```

```
1  def infixe(self, position: int, parcours: list) -> None:
2      if position < len(self.arbre) and \
3                      self.arbre[position] is not None:
4          self.infixe(2*position+1, parcours)
5          parcours.append(self.arbre[position])
6          self.infixe(2*position+2, parcours)
7
8
9
10
11 def postfixe(self, position: int, parcours: list) -> None:
12     if position < len(self.arbre) and \
13                     self.arbre[position] is not None:
14         self.postfixe(2*position+1, parcours)
15         self.postfixe(2*position+2, parcours)
16         parcours.append(self.arbre[position])
```

```
1  def prefixe_2(self, position: int) -> list:
2      if position < len(self.arbre) and \
3                      self.arbre[position] is not None:
4          return [self.arbre[position]] + \
5                      self.prefixe_2(2*position+1) + \
6                      self.prefixe_2(2*position+2)
7      else:  # cas limite
8          return []
```

Remarque

La méthode retourne un tableau, il faut donc retourner un tableau vide dans le cas limite.