

7.3

Algorithme des arbres binaires

NSI TERMINALE - JB DUTHOIT

Notations pour les algorithmes : Soit T un arbre :

T.racine est le nœud racine de l'arbre T

Soit un nœud x :

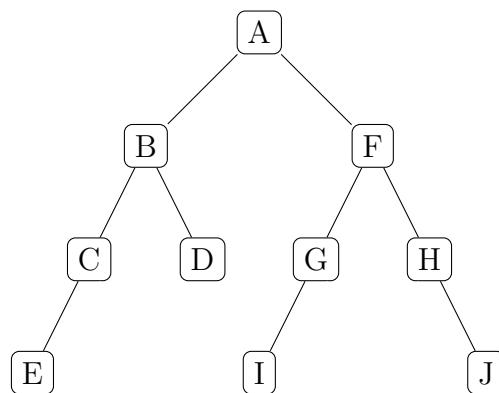
x.gauche correspond au sous-arbre gauche du nœud x

x.droit correspond au sous-arbre droit du nœud x

x.cle correspond à la clé du nœud x

7.3.1 Calcul de la taille d'un arbre

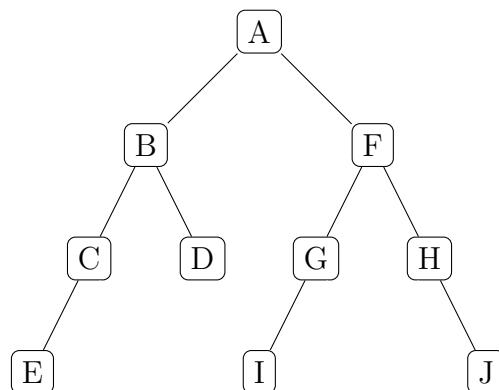
On considère de nouveau cet arbre :



Exercice 7.16

Créer une fonction récursive python `taille(a)` qui renvoie le nombre de nœuds de l'arbre binaire `a`.

7.3.2 Calcul de la hauteur d'un arbre



Exercice 7.17

Créer une fonction récursive python `hauteur(a)` qui renvoie la hauteur de l'arbre binaire `a`.
 ➔ On pourra utiliser la fonction native `max` de python.

7.3.3 Parcours d'un arbre binaire

Un peu de vocabulaire ;-)

Blanchâtre : hâtre est un suffixe à Blanc

préhistoire : pré est un préfixe du mot histoire

sautiller : ill est un infixe dans le mot sauter

On veut ici afficher les différentes valeurs de contenues dans tous les n œuds de l'arbre, par exemple une par ligne. L'ordre dans lequel est effectuée la lecture est donc très important. Il y a deux types de parcours :

- Le **parcours en profondeur** : cela consiste, pour un arbre non vide, à le parcourir récursivement : on parcourt son sous-arbre gauche, puis son sous-arbre droit, sa racine pouvant être traitée au début , entre les deux parcours ou à la fin . On distingue donc 3 parcours en profondeur :
 - **Parcours préfixe** : on traite la racine, puis le sous-arbre gauche, puis le sous-arbre droit.
 - **Parcours infixé** : on traite le sous-arbre gauche, puis la racine, puis le sous-arbre droit.
 - **Parcours postfixé** : on traite le sous-arbre gauche, puis le sous-arbre droit, puis la racine.
- Le **parcours en largeur** : le parcours en largeur d'un arbre binaire non vide consiste à le parcourir par niveaux.

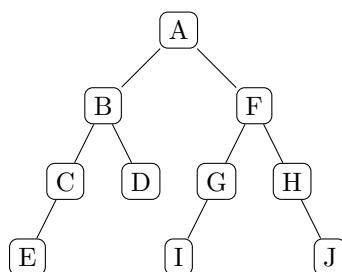
Parcours de l'arbre avec un traitement infixé

Définition

Un **parcours infixé** fonctionne comme suit : On parcourt le sous arbre de gauche, puis on affiche sa racine, et enfin on parcourt le sous arbre droit.

Exercice 7.18

On considère l'arbre binaire suivant :



Sans utiliser d'algorithme ou de programme, donner l'ordre d'affichage des étiquettes pour un parcours

· infixe.

Voici un algorithme permettant un parcours infixe :

```

1 VARIABLE
2 T : arbre
3 x : noeud
4 DEBUT
5 Function PARCOURS_INF(T)
6   if T ≠ NIL then
7     x ← T.racine
8     PARCOURS_INF(x.gauche)
9     affiche x.cle
10    PARCOURS_INF(x.droit)
11  end
12 end

```

Exercice 7.19

l'algorithme de parcours infixe ci-dessus doit être connu et maitrisé. Analysez et comprenez cet algorithme

Exercice 7.20

Créer une fonction python `parcours_infixe(a)` qui effectue un parcours infixe de l'arbre binaire a

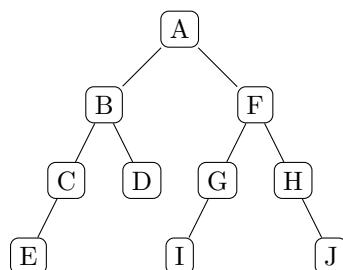
Parcours de l'arbre ordre préfixe

Définition

Un **parcours préfixe** fonctionne comme suit : On affiche la racine, on parcourt le sous arbre gauche, puis le sous arbre droit.

Exercice 7.21

On considère l'arbre binaire suivant :



Sans utiliser d'algorithme ou de programme, donner l'ordre d'affichage des étiquettes pour un parcours préfixe.

Voici un algorithme permettant un parcours préfixe :

```

1 VARIABLE
2 T : arbre
3 x : noeud
4 DEBUT
5 Function PARCOURS_ PREFIXE(T)
6   if T ≠ NIL then
7     x ← T.racine
8     affiche x.cle
9     PARCOURS_ PREFIXE(x.gauche)
10    PARCOURS_ PREFIXE(x.droit)
11  end
12 end

```

Exercice 7.22

l'algorithme de parcours préfixe ci-dessus doit être connu et maitrisé. Analysez et comprenez cet algorithme

Exercice 7.23

Créer une fonction **parcours_prefixe(a)** qui effectue un parcours prefixe de l'arbre binaire a

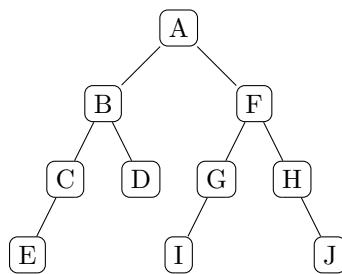
Parcours d'un arbre ordre suffixe

Définition

Un **parcours postfixe** ou **parcours suffixe** fonctionne comme suit : on parcourt le sous arbre gauche, puis le sous arbre droite et on affiche sa racine.

Exercice 7.24

On considère l'arbre binaire suivant :



Sans utiliser d'algorithme ou de programme, donner l'ordre d'affichage des étiquettes pour un parcours suffixe.

Voici un algorithme permettant un parcours suffixe :

```

1 VARIABLE
2 T : arbre
3 x : noeud
4 DEBUT
5 Function PARCOURS_SUFFIXE(T)
6   if T ≠ NIL then
7     x ← T.racine
8     PARCOURS_SUFFIXE(x.gauche)
9     PARCOURS_SUFFIXE(x.droit)
10    afiche x.cle
11  end
12 end
```

Exercice 7.25

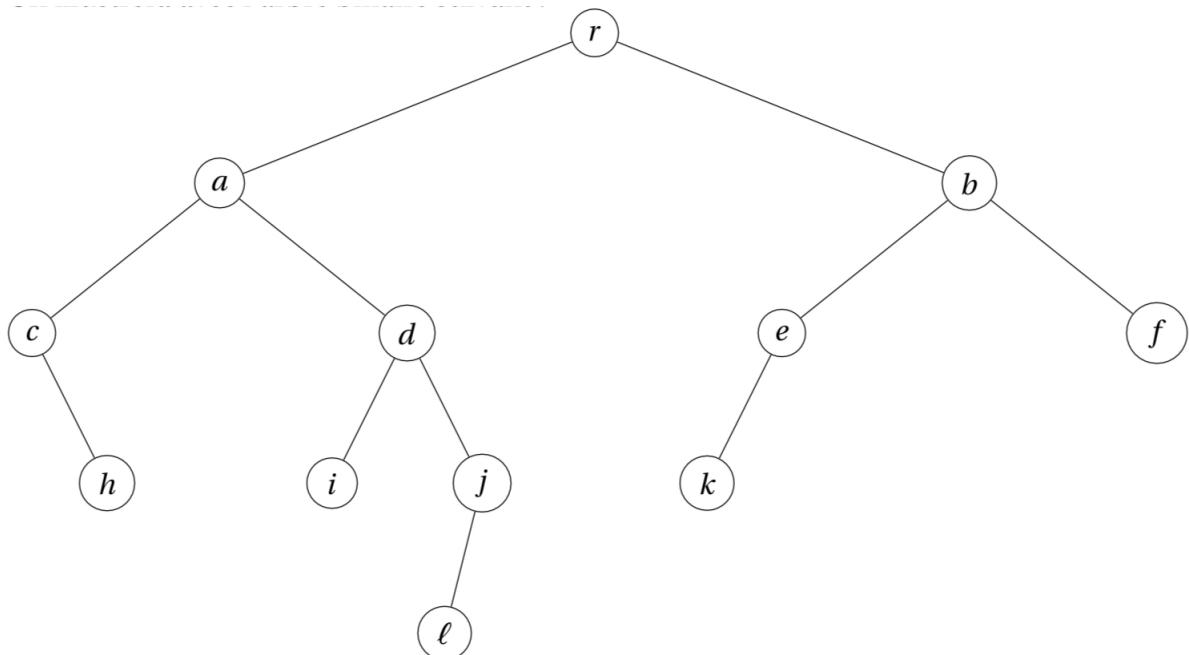
l'algorithme de parcours suffixe ci-dessus doit être connu et maitrisé. Analysez et comprenez cet algorithme

Exercice 7.26

Créer une fonction python `parcours_suffixe(a)` qui effectue un parcours suffixe de l'arbre binaire a.

Exercice 7.27

On considère l'arbre binaire suivant :



Déterminer :

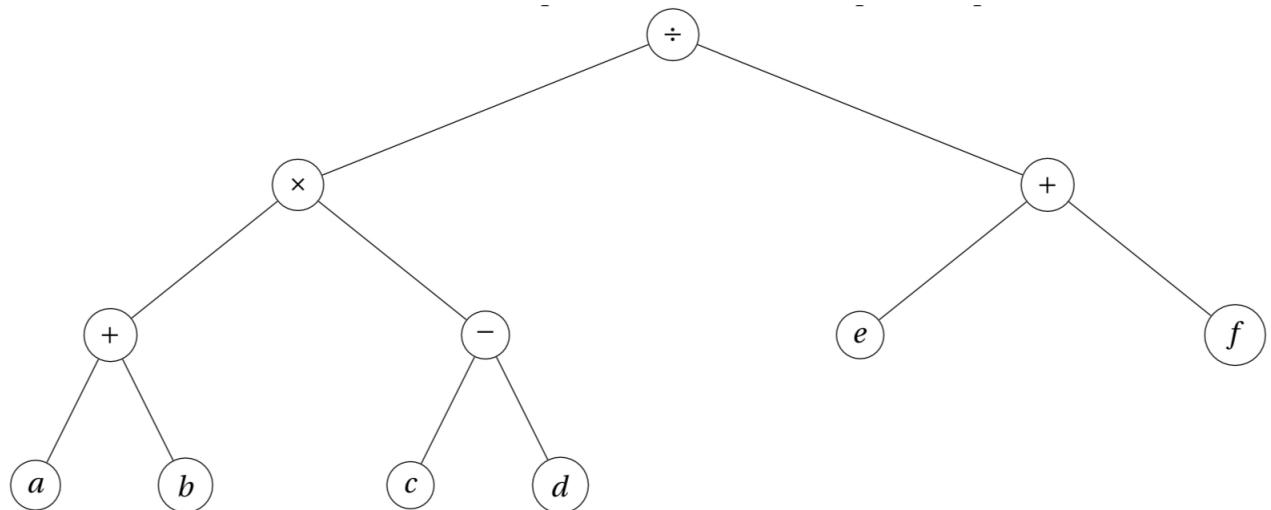
1. La taille de l'arbre
2. La hauteur de l'arbre
3. Son arité
4. L'ordre infixe
5. L'ordre préfixe

6. L'ordre suffixe

Exercice 7.28

Cet exercice porte sur la notation polonaise inverse.

On considère l'arbre binaire suivant :

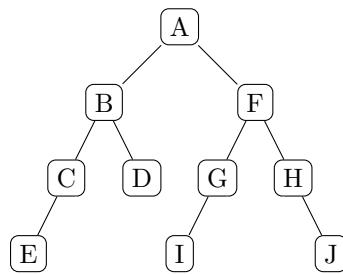


Déterminer :

1. a) La taille de l'arbre
b) La hauteur de l'arbre
c) Son arité
d) L'ordre infixe
e) L'ordre préfixe
f) L'ordre suffixe
2. Pour le parcours infixe, on ajoute la convention suivante : On ajoute une parenthèse ouvrante à chaque fois qu'on entre dans un sous-arbre et on ajoute une parenthèse fermante dès que l'on quitte ce sous arbre. Pour les feuilles, on ne met pas de parenthèses.
 - a) Avec cette convention de parenthèses dans l'ordre infixe, donner l'expression obtenue
→ Cette convention est indispensable pour lever toute ambiguïté. Ce n'est pas le cas pour les deux autres ordres :
 - b) L'ordre préfixe consiste à voir les opérateurs comme fonctions de deux variables, avec la fonction écrite à gauche. Écrire l'expression.
 - c) L'ordre postfixe consiste à voir les opérateurs comme fonctions de deux variables, avec la fonction écrite à droite. Écrire l'expression.

Parcourir un arbre en largeur d'abord**Exercice 7.29**

On considère l'arbre binaire suivant :



Sans utiliser d'algorithme ou de programme, donner l'ordre d'affichage des étiquettes pour un parcours en largeur d'abord.

Voici un algorithme permettant un parcours en largeur d'abord :

```

1 VARIABLE
2 T : arbre
3 Tg : arbre
4 Td : arbre x : noeud
5 f : file
6 Function PARCOURS_LARGEUR(T)
7   result ← tableau vide
8   if T non vide then
9     f ← file vide
10    f.enfiler(T)
11    while f non vide do
12      T_courant = f.defiler()
13      Ajouter racine de T_courant à result
14      if T_courant.gauche non vide then
15        f.enfiler(T_courant.gauche)
16      end
17      if T_courant.droite non vide then
18        f.enfiler(T_courant.droite)
19      end
20    end
21  end
22  Renvoyer result
23 end
  
```

Exercice 7.30

l'algorithme de parcours en largeur ci-dessus doit être connu et maîtrisé. Analysez et comprenez cet algorithme.

Remarque

- Cet algorithme utilise une file FIFO
- Cet algorithme n'est pas récursif.

Exercice 7.31

Créer une fonction **parcours_largeur(a)** qui effectue un parcours en largeur d'abord de l'arbre binaire a