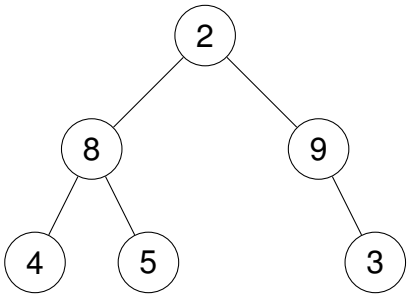
Travaux pratiques - Structures hiérarchiques : arbres

Exercice 1 Algorithmes sur les arbres binaires

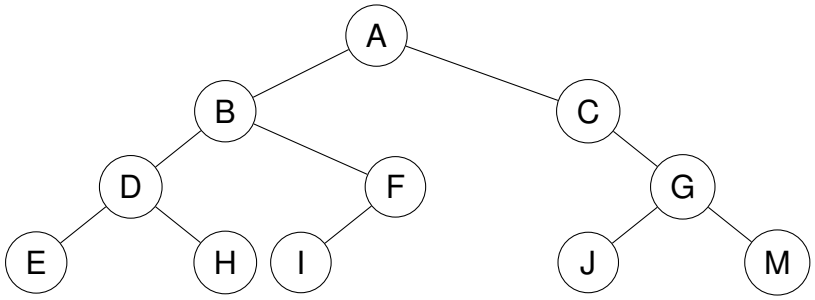
Ci-dessous une implémentation Python d'une classe `Noeud` :

```
class Noeud:
    def __init__(self, e, g=None, d=None):
        self.etiquette = e
        self.gauche = g
        self.droit = d

    def est_feuille(self):
        return not self.gauche and not self.droit

    # Une representation possible de l'arbre
    def __repr__(self):
        ch = str(self.etiquette)
        if self.gauche or self.droit:
            ch = ch + '-(' + str(self.gauche) + ',' + str(self.droit) + ')'
        return ch
```

On considère les arbres `A1` et `A2` suivant :



Arbre `A1`



Arbre `A2`

1. Ecrire les instructions python permettant de créer les arbres `A1` et `A2`.
2. (a) Ecrire une fonction python `taille`, prenant en paramètre un objet `a` de type `Noeud`, qui renvoie la taille de l'arbre `a`.
   (b) Ecrire sur papier la pile des appels récursifs lors de l'appel de la fonction `taille(A1)`.
   (c) Tester la fonction `taille` avec l'arbre `A1`.
3.
4. (a) Ecrire une fonction python `hauteur`, prenant en paramètre un objet `a` de type `Noeud`, qui renvoie la hauteur de l'arbre `a`.
   (b) Ecrire sur papier la pile des appels récursifs lors de l'appel de la fonction `hauteur(A1)`.
   (c) Tester la fonction `hauteur` avec l'arbre `A1`.
5. (a) Ecrire une fonction python `parcours_prefixe`, prenant en paramètre un objet `a` de type `Noeud`, qui affiche la liste des étiquettes de l'arbre `a` selon le parcours en profondeur prefixe.
   (b) Pour chacun des arbres `A1` et `A2`, donner la liste des étiquettes lors d'un parcours en profondeur prefixe.
   (c) Tester votre fonction avec les arbres `A1` et `A2` et comparer avec le résultat attendu.
6. (a) Ecrire une fonction python `parcours_infixe`, prenant en paramètre un objet `a` de type `Noeud`, qui affiche la liste des étiquettes de l'arbre `a` selon le parcours en profondeur infixe.

(b) Pour chacun des arbres `A1` et `A2`, donner la liste des étiquettes lors d'un parcours en profondeur infixe.

(c) Tester votre fonction avec les arbres `A1` et `A2` et comparer avec le résultat attendu.

7. (a) Ecrire une fonction python `parcours_suffixe`, prenant en paramètre un objet `a` de type `Noeud`, qui affiche la liste des étiquettes de l'arbre `a` selon le parcours en profondeur suffixe.

(b) Pour chacun des arbres `A1` et `A2`, donner la liste des étiquettes lors d'un parcours en profondeur suffixe.

(c) Tester votre fonction avec les arbres `A1` et `A2` et comparer avec le résultat attendu.

Exercice 2 Recherche dans un arbre binaire

On veut écrire une fonction récursive `etq_presente(A, e)` qui renvoie `Vrai` si l'étiquette `e` est présente dans l'arbre binaire `A` et `Faux` sinon.

1. Recopier et compléter la phrase suivante présentant le principe de l'algorithme de la recherche :

   Si l'arbre est ... alors on renvoie ..., sinon on regarde si l'étiquette est ... ou ... ou ....

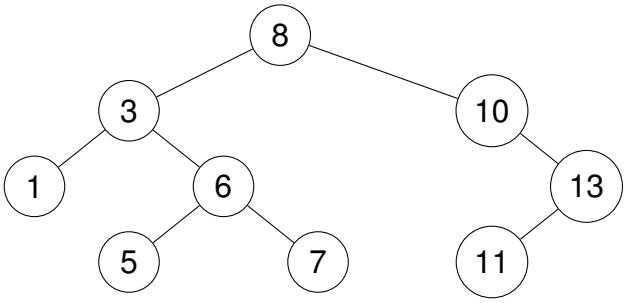
2. Ecrire la fonction récursive etq_presente(A, e). On suppose que les AB sont implémentés avec la classe Noeud (voir exercice précédent).
3. Proposer un bon jeu de test à l'aide des arbres `A1` et `A2` de l'exercice précédent et vérifier que votre fonction passe tous les tests avec succès.
4. Quel est le coût en temps de cet algorithme (dans le pire cas) ?

Exercice 3 Algorithme de recherche dans un ABR

On veut écrire une fonction récursive `etq_presente(A, e)` qui renvoie `Vrai` si l'étiquette `e` est présente dans l'ABR `A`, et `Faux` sinon.

1. Recopiez et compléter l'algorithme de cette fonction.

```
fonction etq_presente(A, e)  ->
   Boolean
    si est_vide(A) alors
        ...
    sinon
        si ... alors
            ...
        sinon
            si ... alors
                ...
            sinon
                ...
            fin si
        fin si
    fin si
```



Arbre `A3`

2. Implémenter cette fonction en Python. (utiliser la représentation objet des exercices précédents).
3. Tester la fonction en utilisant l'ABR `A3` donné ci-contre.

Exercice 4 Caractérisitiques d'un ABR

Dans cet exercice on considère l'ABR `A3` défini dans l'exercice précédent.

Ordre de parcours

1. Donner l'ordre des noeuds visités selon les parcours par ordre préfixe, infixe et suffixe.
2. Que remarque-t-on ?

Recherche du minimum

1. Rappeler comment trouver l'étiquette minimale d'un ABR.
2. Ecrire une fonction itérative `etq_min(A)` qui renvoie l'étiquette minimale d'un ABR `A` non vide.
3. Ecrire une fonction récursive `etq_min_rec(A)` qui renvoie l'étiquette minimale d'un ABR `A` non vide.
4. Tester les deux fonctions avec l'arbre `A3`.

Recherche du maximum

1. Rappeler comment trouver l'étiquette maximale d'un ABR.
2. Ecrire une fonction itérative `etq_max(A)` qui renvoie l'étiquette maximale d'un ABR `A` non vide.
3. Ecrire une fonction récursive `etq_max_rec(A)` qui renvoie l'étiquette maximale d'un ABR `A` non vide.
4. Tester les deux fonctions avec l'arbre `A3`.

Exercice 5 Insérer une clé dans un ABR

Ce dernier exercice traite l'insertion d'une clé dans un ABR, un algorithme au programme. Dans un premier temps, vous écrirez un algorithme qui fait l'insertion avec modification en place de l'ABR, et dans un second temps vous écrirez une version plus simple mais qui renvoie un nouvel arbre à chaque insertion (et ne modifie donc pas les arbres passés en argument).
Dans cet exercice on considère l'ABR `A3` défini dans l'exercice 3.

Version avec modification en place

1. On veut insérer l'étiquette "4" dans l'ABR `A3`. Expliquez comment procéder en partant de la racine.
2. Même question pour insérer l'étiquette "2".
3. Même question pour insérer l'étiquette "9".
4. Recopier et compléter le principe de l'algorithme récursif suivant qui ajoute la clé `e` dans l'ABR non vide `A`.

```
fonction ajouter(A, e)
    si e <= etiquette(A) alors
        si est_vide(gauche(A)) alors
            ...
        sinon
            ajouter(...)
        fin si
    sinon
        ...
```

5. Implémenter cette fonction en Python. (utiliser la représentation d'un ABR par la classe Noeud).
6. Tester votre fonction en insérant les clés 4, 2 puis 9. Vérifier ensuite que les clés ont bien été insérées aux bons endroits (en affichant l'arbre).
7. Créer un ABR à un noeud. Ecrire ensuite les instructions permettant d'y insérer 4 clés en affichant l'arbre modifié à chaque insertion pour vérifier.

> **Remarque** : L'algorithme d'insertion ainsi écrit doit se faire dans un ABR non vide. En effet, le cas de l'insertion dans un arbre vide (représenté par None) n'est pas pris en compte. On pourrait le faire mais cela compliquerait les choses. En revanche, si on écrit un algorithme qui ne modifie pas en place l'arbre passé en argument on peut écrire un algorithme d'insertion plus court et qui englobe le cas de l'insertion dans un arbre vide.

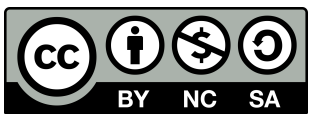
Version avec construction de nouveaux arbres

Pour terminer, on va écrire une fonction `ajouter(A, e)` qui renvoie un nouvel arbre contenant `e` et tous les noeuds de l'arbre `A`. On va encore procéder récursivement.

1. Le cas de base concerne l'ajout dans un arbre vide. Que doit renvoyer la fonction dans ce cas ?
2. Si l'abre `A` n'est pas vide, il faut comparer `e` à l'étiquette de `A`. Selon le cas, il faut ajouter `e` au sous-arbre gauche ou au sous-arbre droit de `A`.
   Compléter avec des phrases ce que la fonction doit faire selon les cas :

```
si e <= etiquette(A) alors
    renvoyer Noeud(....................., .....................,
       .....................)
sinon
    renvoyer Noeud(....................., .....................,
       .....................)
```

3. Ecrire cette fonction récursive en Python. N'oubliez pas le cas de base.
4. Tester votre fonction en insérant les clés 4, 2 puis 9. Vérifier ensuite que les clés ont bien été insérées aux bons endroits (en affichant l'arbre).
5. Ecrire ensuite les instructions permettant de construire le même ABR que dans la question 7 en procédant par insertion successives.



Source : Lycée Mounier - Angers