Séq. 8 – Les arbres binaires de recherche

Objectifs

1. Distinguer les arbres binaires
2. Parcourir un arbre de différentes façons (ordres infixe, préfixe ou suffixe ; ordre en largeur d'abord)
3. Rechercher une clé dans un arbre de recherche, insérer une clé

Cours inspiré des pages :
https://grug.eeisti.fr/ING-1/Info%20/Algorithmique/TD/td7Corrige.pdf
https://lyceum.fr/tg/nsi/5-algorithmique/1-algorithmes-sur-les-arbres-binaires

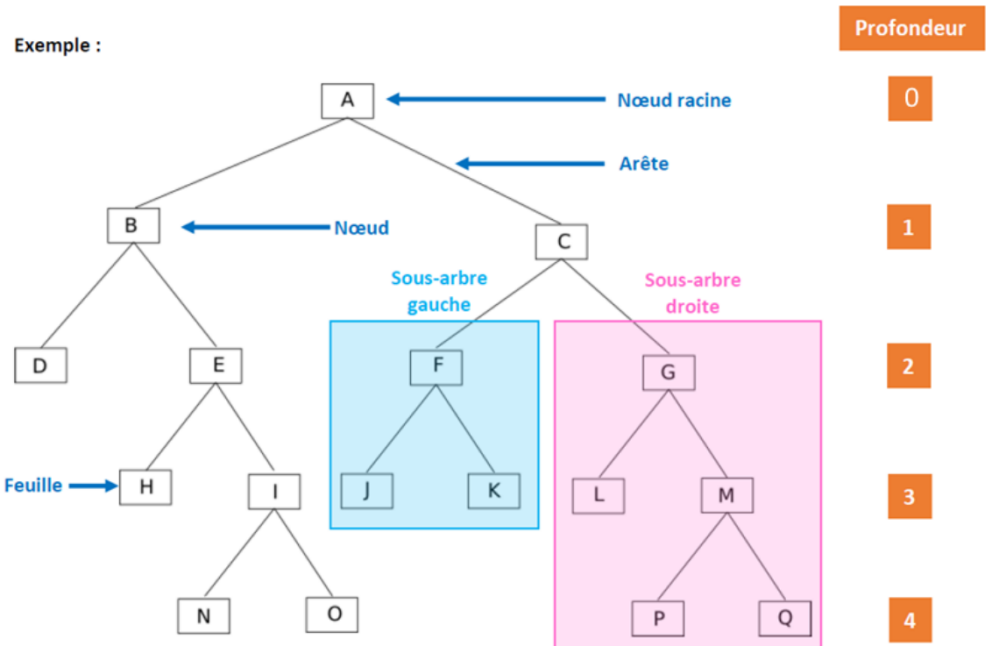
1 Définition `Arbre binaire`

Un arbre binaire est un arbre d'arité deux.
Un arbre binaire est :

- soit l'arbre vide, noté Δ ;
- soit un triplet (e, g, d), appelée nœud, dans lequel
- e est l'élément, ou encore étiquette, de la racine de l'arbre,
- g est le sous-arbre gauche de l'arbre
- d est le sous-arbre droit de l'arbre.

Les sous-arbres gauche et droit d'un arbre binaire non vide sont eux-mêmes des arbres binaires. La structure d'arbre binaire est donc une structure récursive.
On appelle fils gauche, respectivement fils droit, le sous-arbre gauche, respectivement droit, d'un nœud.

A faire vous même 1.



Quelle est la taille de cet arbre ? Quel est sa hauteur ? Combien de feuilles ? Combien de nœuds internes ? Quels sont les fils du nœud B ? Quelles sont les profondeurs de H, J et P ? Quelle est la racine de l' arbre gauche ? De l' arbre droit ?

2 Activités d’ appropriation du vocabulaire

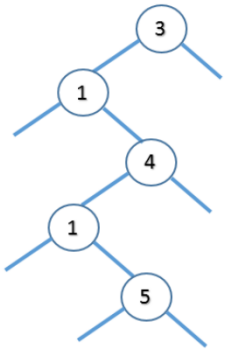
2.1 Dessins d’ arbres binaires

A faire vous même 2.

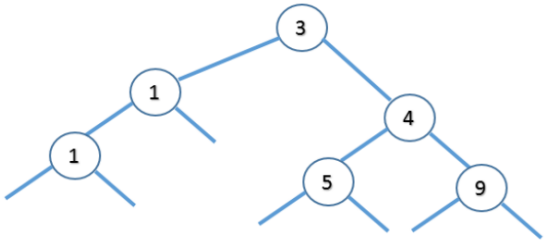
Pour chaque description, dessinez le schéma du graphe (rappel : arbre vide est noté Δ). Précisez la taille, la hauteur et le nombre de feuilles :

- Cas1 : (1, Δ, Δ) :

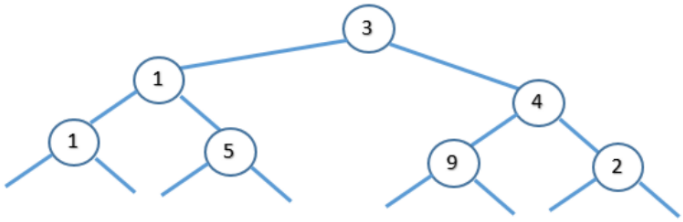
- (3, (1, Δ, (4, (1, Δ, (5, Δ, Δ)), Δ)), Δ)



- Cas3 : (3, (1, (1, Δ, Δ), Δ), (4, (5, Δ, Δ), (9, Δ, Δ)))



- Cas4 : (3, (1, (1, Δ, Δ), (5, Δ, Δ)), (4, (9, Δ, Δ), (2, Δ, Δ)))



2.2 Caractéristiques d’ arbres binaires (avec une racine de profondeur 0)

A faire vous même 3.

1. Combien de feuilles au minimum comporte un arbre binaire de hauteur h ? Et au maximum ?
2. Combien de nœuds au minimum comporte un arbre binaire de hauteur h ? Et au maximum ?

2.3 Squelette d’ arbre binaire

Le squelette (fig.2) d’un arbre (fig.1) définit sa géométrie: on obtient ce squelette en supprimant toute information aux nœuds et en supprimant les feuilles.

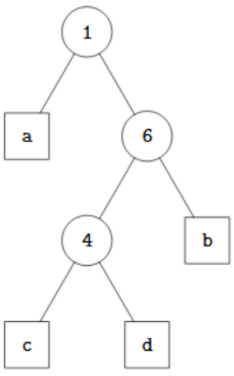


Figure 1

Figure 2 Son squelette

3. Combien y a-t-il de squelette d' arbre binaire de taille 1 ? Dessinez-les.
4. Combien y a-t-il de squelette d' arbre binaire de taille 2 ? Dessinez-les.
5. Combien y a-t-il de squelette d' arbre binaire de taille 3 ? Dessinez-les.
6. Combien y a-t-il de squelette d' arbre binaire de taille 4 ? Dessinez-les.

3 Python : Programmation objet et arbres binaires

A faire vous même 4.

1. Écrivez une classe `Noeud` (pour arbre binaire) en python.
2. Reprenez l' arbre binaire du A faire vous-même 1 et construisez-le à l' aide de cette classe.

4 Algorithmes : Calcul des taille et hauteur

Les arbres binaires étant des structures récursives, les algorithmes associés sont très souvent des algorithmes récursifs.

A faire vous même 5.

Écrivez l' algorithme de calcul de la taille d' un arbre binaire.

Proposition 1 :

```
- Si l'arbre est vide :
    renvoyer taille = 0
- Sinon
    renvoyer taille=1 + la somme du nbre de nœuds des ss-arbres gauche et droit.
```

Proposition 2 :

```
fonction NombreNoeud(T : arbre ) # renvoie un entier
    si (EstVide(T))
        renvoyer 0;
    sinon
        renvoyer 1 + NombreNoeud(FilsGauche(T)) + NombreNoeud(FilsDroit(T));
    fin si
```

A faire vous même 6.

Écrivez l' algorithme de calcul de la hauteur d' un arbre binaire.

Proposition 1 :

```
- Si l'arbre est vide :
    renvoyer hauteur = -1
- Sinon
    renvoyer hauteur = 1 + la hauteur maximale entre ses fils.
```

Proposition 2 :

```
fonction hauteur (T : arbre) renvoie un entier
    si T est vide
        renvoyer -1
    sinon
        renvoyer 1 + max(hauteur (FilsGauche(T)) , hauteur(FilsDroit(T)))
```

```
        fin si
```

A faire vous même 7.

P. 132 ex 8
Prenez la description des arbres à l' aide de tuples de tuples (avec Δ = None).
Écrivez un programme python qui reprend les 2 algorithmes ci-dessus pour calculer taille et hauteur.

A faire vous même 8.

Reprenez la programmation de la class `BinTree` P.124 du livre.
Recopiez la class et la méthode qui calcule la taille de l' arbre
P. 132 ex 7, écrivez un programme python qui reprend l' algorithme ci-dessus pour calculer la hauteur, le nombre de feuilles, ...

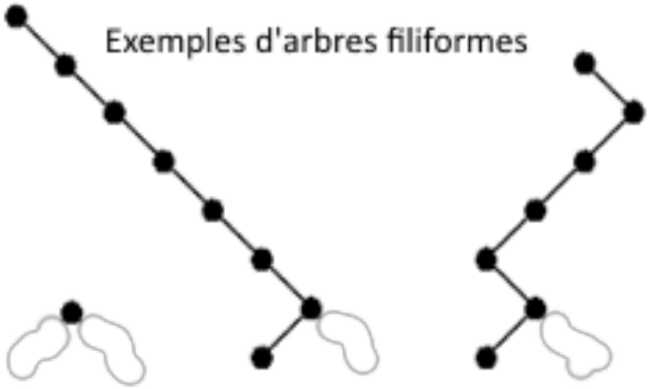
Lire P. 124-125

5 Arbres binaires particuliers

Les arbres binaires sont caractérisés par le fait que chaque nœud possède au plus deux fils.
D'autres caractéristiques sont définies, qui permettent par exemple d'identifier des arbres pour lesquels le coût de certaines opérations sera minimal, ou de définir des algorithmes spécifiques à ces arbres.

5.1 Arbre binaire filiforme

Un arbre binaire filiforme ou dégénéré est un arbre dans lequel tous les nœuds internes n'ont qu'un seul fils (Un arbre filiforme ne possède donc qu'une unique feuille).



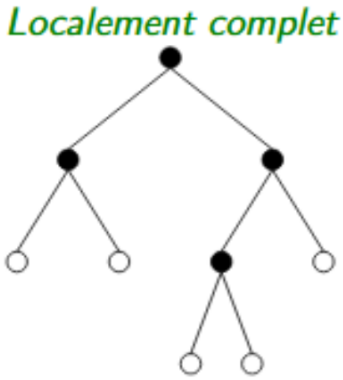
A faire vous même 9.

Écrivez l' algorithme permettant de reconnaître un arbre binaire filiforme

```
Fonction binaireFil(A : ArbreBinaire) : Booléen
    Début
    Si hauteur(A) <=0 Alors
        retourner faux
    SinonSi not(estVide(filsGauche(A))) et not(estVide(filsDroit(A)))
        retourner faux
    SinonSi estVide(filsDroit(A))
        Si hauteur(A)=1
            Retourner vrai
        sinon
            retourner binaireFil(filsGauche(A))
    SinonSi estVide(filsGauche(A))
        Si hauteur(A)=1
            Retourner vrai
        Sinon
            Retourner binaireFil(filsDroit(A))
        FinSi
    Fin
```

5.2 Arbre binaire localement complet

Un arbre binaire localement complet ou arbre binaire strict est un arbre dont tous les nœuds internes possèdent exactement deux fils. (Autrement dit, un arbre binaire localement complet est un arbre dont chaque nœud possède zéro ou 2 fils. L'arbre vide n'est pas localement complet.)



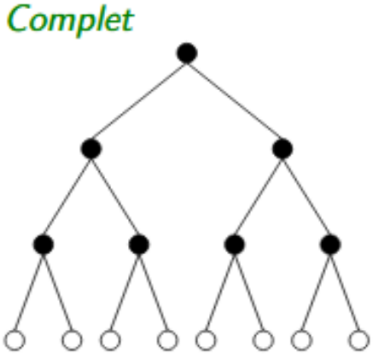
A faire vous même 10.

Écrivez l' algorithme permettant de reconnaître un arbre binaire localement complet

```
Fonction localementComplet(A : ArbreBinaire) : Booléen
    Début
    Si estVide(A) Alors
        retourner vrai
    Sinonsi estVide(filsGauche(A)) et estVide(filsDroit(A))
        retourner vrai
    Sinonsi estVide(filsGauche(A)) ou estVide(filsDroit(A))
        retourner faux
    Sinon
        retourner localementComplet(filsGauche(A)) et localementComplet(filsDroit(A))
    FinSi
    Fin
```

5.3 Arbre binaire complet

Un arbre binaire complet est un arbre binaire localement complet dans lequel toutes les feuilles sont à la même profondeur (Il s'agit d'un arbre dont tous les niveaux sont remplis).



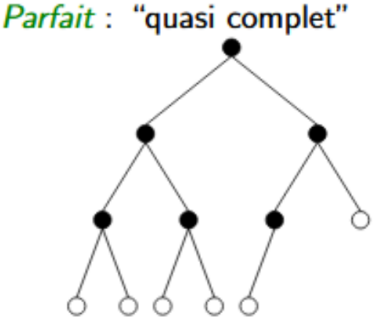
A faire vous même 11.

Écrivez l' algorithme permettant de reconnaître un arbre binaire complet

```
Fonction Complet(A : ArbreBinaire) : Booléen
    Début
    Si estVide(A)
        retourner vrai
    SinonSi hauteur(filsGauche(A)) != hauteur(filsDroit(A))
        retourner faux
    Sinon
        Retourner complet(filsGauche(A)) et complet(filsDroit(A))
    FinSi
    Fin
```

5.4 Arbre binaire parfait

Un arbre binaire parfait est un arbre dans lequel tous les niveaux sont remplis à l'exception éventuelle du dernier, et dans ce cas les feuilles du dernier niveau sont alignées à gauche.

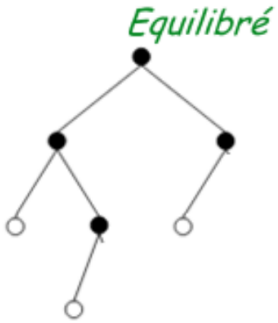


A faire vous même 12.

Écrivez l' algorithme permettant de reconnaître un arbre binaire parfait

```
Fonction parfait(A : ArbreBinaire) : Booléen
    Début
    Si estVide(A) Alors
        retourner vrai
    SinonSi hauteur(filsGauche(A)) = hauteur(filsDroit(A))
        retourner complet(filsGauche(A)) et parfait(filsDroit(A))
    SinonSi hauteur(filsGauche(A)) = hauteur(filsDroit(A+ 1))
        retourner parfait(filsGauche(A)) et complet(filsDroit(A))
    Sinon
        retourner faux
    FinSi
    Fin
```

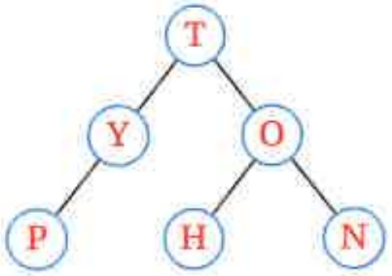
5.5 Arbre binaire équilibré



Un arbre binaire équilibré est un arbre dont les deux fils sont des arbres équilibrés dont les hauteurs diffèrent d'au plus une unité. Ainsi, l'accès à n'importe lequel des nœuds est en moyenne minimisé.

6 Parcours de l' arbre binaire

6.1 Parcours préfixe



Dans cet ordre, chaque nœud est visité puis chacun de ses fils.
Voici le pseudo-code extrait de l'article Wikipedia sur les arbres.

```
parcours_préfixe(Arbre A) :
    visiter (A)
    Si nonVide (gauche(A))
        parcours_préfixe(gauche(A))
    Si nonVide (droite(A))
        parcours_préfixe(droite(A))
```

Pour l' arbre ci-dessus, le résultat est : T – Y – P – O – H - N

A faire vous même 13.

Reprenez la programmation de la class `BinTree` P.124 du livre.
Écrivez un programme python qui reprend l' algorithme ci-dessus pour établir la liste des nœuds lors d' un parcours préfixe de l' arbre.

6.2 Parcours infixe

On visite chaque nœud entre les nœuds de son sous-arbre de gauche et les nœuds de son sous-arbre de droite. C'est une manière assez commune de parcourir un arbre binaire de recherche, car il donne les valeurs dans l'ordre croissant.
Voici le pseudo-code extrait de l'article Wikipedia sur les arbres.

```
parcours_infixe(Arbre A) :
    Si nonVide(gauche(A))
       parcours_infixe(gauche(A))
    visiter(A)
    Si nonVide(droite(A))
       parcours_infixe(droite(A))
```

Pour l' arbre ci-dessus, le résultat est : P – Y – T – H – O - N

A faire vous même 14.

Reprenez la programmation de la class `BinTree` P.124 du livre.
Écrivez un programme python qui reprend l' algorithme ci-dessus pour établir la liste des nœuds lors d' un parcours infixe de l' arbre.

6.3 Parcours postfixe

On affiche chaque nœud après avoir affiché chacun de ses fils.
Voici le pseudo-code extrait de l'article Wikipedia sur les arbres.

```
parcours_postfixe(Arbre A) :
    Si nonVide(gauche(A))
       parcours_postfixe(gauche(A))
    Si nonVide(droite(A))
       parcours_postfixe(droite(A))
    visiter(A)
```

Pour l' arbre ci-dessus, le résultat est : P – Y – H – N – O - T

A faire vous même 15.

Reprenez la programmation de la class `BinTree` P.124 du livre.
Écrivez un programme python qui reprend l' algorithme ci-dessus pour établir la liste des nœuds lors d' un parcours postfixe de l' arbre.

6.4 Parcours en largeur

On parcours les noeuds de gauche à droite étage par étage, comme si on «lisait» l'arbre.
Voici le pseudo-code extrait de l'article Wikipedia sur les arbres.

```
parcours_largeur(Arbre A)
   f = FileVide
   enfiler(Racine(A), f)
   Tant que (f != FileVide)
       nœud = defiler(f)
       Visiter(nœud)                          // On fait une opération
       Si (gauche(nœud) != null) Alors
           enfiler(gauche(nœud), f)
       Si (droite(nœud) != null) Alors
           enfiler(droite(nœud), f)
```

Pour l' arbre ci-dessus, le résultat est : T – Y – O – P – H - N

A faire vous même 16.

Reprenez la programmation de la class `BinTree` P.124 du livre.
Écrivez un programme python qui reprend l' algorithme ci-dessus pour établir la liste des nœuds lors d' un parcours en largeur de l' arbre.

Lire P. 122-123

P. 130 ex 2

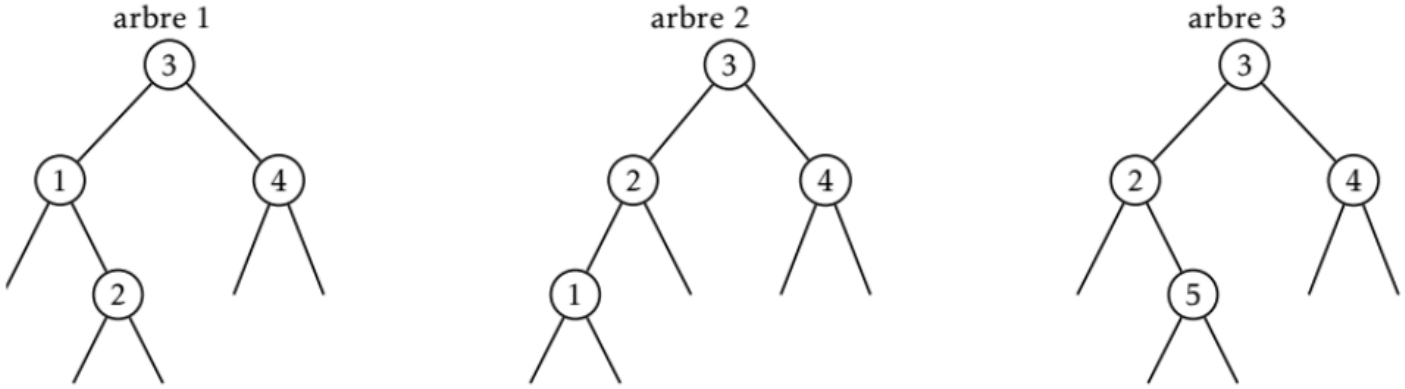
P. 132 ex 6

7 Utilisation des arbres binaires

7.1 Arbres binaires de recherche

7.1.1 Définition

Il s'agit d'un arbre binaire dans lequel toutes les valeurs dans le sous-arbre gauche d'un nœud sont inférieures à la valeur à la racine de l'arbre et toutes les valeurs dans le sous-arbre droit d'un nœud sont supérieures ou égales à la valeur à la racine de l'arbre.



Les arbres 1 et 2 sont bien des ABR mais pas le 3, en effet 5 est dans le sous arbre de gauche de la racine 3 ce qui est impossible !
Rq : comme l'illustrent les deux arbres de gauche, les mêmes valeurs peuvent être stockées dans plusieurs ABR différents.

7.1.2 Intérêt – Recherche d' une clé

Le caractère trié d'un arbre binaire de recherche permet des opérations rapides pour rechercher une clé.
La compléxité de l' algorithme de recherche (ou temps de calcul) est linéaire (ou fonction de h, la hauteur de l' arbre) voire moins dans certains cas.

P. 133 ex 12

7.1.3 Insertion d' une clé

L'insertion d'un nœud commence par une recherche : on cherche la clé du nœud à insérer ; lorsqu'on arrive à une feuille, on ajoute le nœud comme fils de la feuille en comparant sa clé à celle de la feuille : si elle est inférieure, le nouveau nœud sera à gauche ; sinon il sera à droite.

```
fonction insertion(a: ABR, clé: élément)
        Si a vide
            retourner ABR(clé, vide, vide)
    Sinon
        e = étiquette de a
        Si e < clé
            retourner ABR(e, insertion(gauche(a), clé), droite(a))
        Sinon
            retourner ABR(e, gauche(a), insertion(droite(a), clé))
```

A faire vous même 17.

Reprenez la programmation de la class `BinTree` P.124 du livre.
Écrivez un prog. python qui reprend l' algorithme ci-dessus pour insérer une clé dans un arbre binaire de recherche.

7.2 Arbre représentant les expressions arithmétiques

Lire P. 127
Reproduire l' exemple donné :

P. 134 ex 14