# Algorithmique 1

# DS 4

Documents **non** autorisés
Remarque la notation dépendra de la lisibilité de la copie.

## Exercice 4.1

1. A partir d'un dictionnaire vide, dessiner le changement d'état du dictionnaire après l'ajout des mots "bonjour", "journalier", "jour", "bonsoir", "bonnet", "journal" et "joueur".
2. Mettre à jour le dictionnaire suite à la suppression des mots "bonjour" et "jour".
3. Écrire une fonction qui renvoie le mot le plus long d'un dictionnaire, en cas d'égalité, renvoie le premier trouvé.

## Exercice 4.2

1. Partant d'un arbre binaire vide, construire un AVL en utilisant la fonction d'insertion (vue en cours) appliquée à la suite des clés suivantes :

   $$9, 12, 14, 17, 19, 23, 50, 54, 67, 72, 78$$

Attention Détailer chaque étape de construction

2. Dans l'AVL obtenu ci-dessus, supponsons qu'on supprime la clé 72, que devient l'arbre résultant ?

## Exercice 4.3

1. Ecrire une fonction qui vérifie si un arbre binaire est un AVL. Par exemple pour la figure(a) et la figure(b) la fonction renvoie `faux` et `vrai` pour la figure(c).

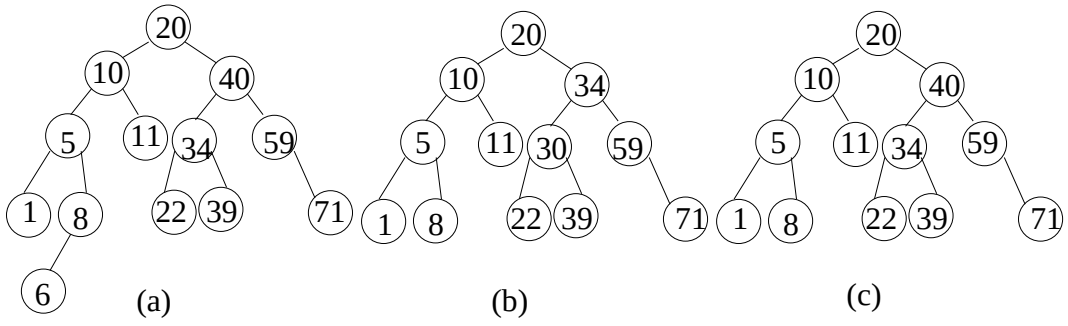


FIG. 1 – Définition

2. Quelle est la complexité de votre fonction ? Justifier votre réponse.

Annexe A : Type abstrait arbreBinaire.

```
arbreBinaire= curseur;
sommet= curseur;
```

Primitives.

- Accès `fonction valeur(val S:sommet):objet;`
  /* vaut NIL si le sommet n'existe pas /*
  `fonction filsGauche(val S:sommet):sommet;`
  /* vaut NIL si S n'a pas de fils gauche */
  `fonction filsDroit(val S:sommet):sommet;`
  /* vaut NIL si S n'a pas de fils droit */
  `fonction pere(val S:sommet):sommet;` /* vaut NIL si S est la racine de l'arbre */
- Modification `fonction setValeur(ref S:sommet, val x:objet):vide;`
  /* affecte au sommet S la valeur x */
  `fonction ajouterFilsGauche(ref S:sommet, val x:objet):vide;`
  /* filsGauche(S)==NIL doit etre verifie */
  `fonction ajouterFilsDroit(ref S:sommet, x:objet):vide;`
  /* filsDroit(S)==NIL doit etre verifie */
  `fonction supprimerFilsGauche(ref S:sommet):vide;`
  /* filsGauche(S) est une feuille */
  `fonction supprimerFilsDroit(ref S:sommet):vide;`
  /* filsDroit(S) est une feuille */
  `fonction detruireSommet(ref S:sommet):vide;`
  /* S est une feuille */
- Création `fonction creerArbreBinaire(val Racine:objet):sommet;`

```
arbreBinaire=curseur;
sommet=curseur;
```

Le type sommet présente les primitives suivantes :

- accès

```
fonction getValeur(val S:sommet):objet;
/* vaut NIL si le sommet n'existe pas /*
fonction filsGauche(val S:sommet):sommet;
fonction filsDroit(val S:sommet):sommet;
fonction pere(val S:sommet):sommet;
```

- modification

```
fonction setValeur(ref S:sommet;val x:objet):vide;
/* affecte au sommet S la valeur x */
fonction ajouterFilsGauche(ref S:sommet,val x:objet):vide;
/* filsGauche(S)==NIL doit être vérifié */
fonction ajouterFilsDroit(ref S:sommet,x:objet):vide;
/* filsDroit(S)==NIL doit être vérifié */
fonction supprimerFilsGauche(ref S:sommet):vide;
/* filsGauche(S) est une feuille */
fonction supprimerFilsDroit(ref S:sommet):vide;
/* filsDroit(S) est une feuille */
fonction detruireSommet(ref S:sommet):vide;
/* S est une feuille */
```

- Il faut par ailleurs pouvoir créer la racine d'un arbre on a donc de plus la primitive

```
fonction creerArbreBinaire(val Racine:objet):sommet;
```

## Annexe B : Type abstrait **arbreAVL**

```
type celluleAVL= structure
                   info: objet;
                   hauteur: entier;
                   gauche: sommetAVL;
                   droit: sommetAVL;
                   pere: sommetAVL;
                 finstructure
sommetAVL= ^celluleAVL;
arbreAVL= sommetAVL;

fonction getHauteur(ref s: sommetAVL): entier;
fonction setHauteur(ref s: sommetAVL): entier;
fonction rotationDroite(ref s: sommetAVL): vide;
fonction rotationGauche(ref s: sommetAVL): vide;
fonction ajouter(ref s: sommetAVL, val e: objet): vide;
fonction supprimer(ref s: sommetAVL): booleen;
fonction equilibrerApresInsertion(ref s: sommetAVL, val cote: entier): vide;
fonction equilibrerApresSuppression(ref s: sommetAVL): vide;
```

```
fonction equilibrerUnSommet(ref p,s : sommetAVL): vide;
```

## Annexe C : **Type abstrait Dictionnaire.**

```
fonction appartient(val d: dictionnaire, val M: mot): booleen;
fonction creerDictionnaire(ref d: dictionnaire): vide;
fonction ajouter(ref d: dictionnaire, val M: mot): vide;
fonction supprimer(ref d: dictionnaire, val M: mot): vide;
fonction detruireDictionnaire(ref d: dictionnaire): vide;
```

## Annexe B : **Implémentation du type abstrait Dictionnaire dans le type arbreBinaire.**

```
type dico= structure
             a: sommet; /* l'arbre */
             p: sommet; /* le curseur */
           finstructure
```