L2 Algorithmique des arbres

Durée 2 heures.

Notes de cours, de TD et de TP manuscrites autorisées.

Supports de cours utilisés en amphis autorisés.

- Les différentes parties sont indépendantes et peuvent être traitées dans n'importe quel ordre.
- On veillera à minimiser la complexité des fonctions.
- On pourra écrire des fonctions annexes, si l'utilisation de paramètres supplémentaires est nécessaire.

▸ **Exercice 1. Arbres binaires.** (5 points)
On utilise la structure :

```
typedef struct noeud{
    int val;
    struct noeud *fg, *fd;
} Noeud, *Arbre;
```

1. Écrire une fonction `int nombre2Enfants(Arbre a)` qui renvoie le nombre de nœuds internes de l'arbre `a` ayant exactement deux enfants.
2. Écrire une fonction `int hauteur(Arbre a)` qui renvoie la hauteur de l'arbre `a`.
3. Écrire une fonction `void afficheNiveau(Arbre a,int n)` qui affiche les étiquettes des nœuds de niveau `n` de l'arbre `a`. L'ordre d'affichage est libre.
4. Écrire une fonction `void libere(Arbre * a)` libérant l'espace mémoire alloué sur le tas pour stocker l'arbre `*a`.
5. Écrire une fonction `Arbre copieArbre(Arbre a)` qui renvoie une copie de l'arbre `a`.

   On supposera que l'on dispose d'une fonction `Noeud * alloueNoeud(int n)` *permettant d'allouer un nœud contenant l'entier* `n`.

▸ **Exercice 2. Insertions et suppressions dans différents types d'arbres** (6 points)
Dans tout cet exercice, lors d'une suppression dans un arbre binaire de recherche, on remontera lorsqu'il y aura le choix la valeur maximale du sous-arbre de gauche.

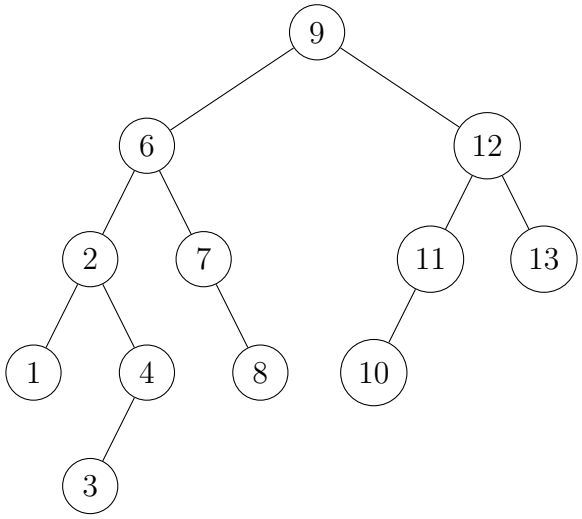
Attention : Les nombres à insérer varient d'une question à l'autre !

1. (a) Dessiner l'arbre binaire de recherche obtenu après insertion dans un arbre initialement vide des nombres 6, 9, 7, 4, 8, 1, 5, 10, 3 et 2.

   (b) Dessiner chaque arbre binaire de recherche obtenus après suppression des nombres 10, 1, 6 et 9 dans l'arbre précédent

2. (a) Dessiner les arbres de type AVL obtenus après les ajouts suivants : 6, 4, 2, 7, 8, 5, 3, 1. *On précisera bien toutes les rotations qui seront effectuées et sur quels nœuds elles sont réalisées ; l'arbre sera dessiné* **avant** *et* **après** *chaque rotation.*

   (b) Dessiner l'arbre AVL obtenu après suppression du nombre 13 dans l'AVL suivant :



3. Dessiner les B-arbres de degré 2 obtenus après les ajouts successifs de 39, 23, 50, 62, 58, 83, 29, 36, 40, 46, 56, 133, 33. *On gèrera les B-arbres de degré 2 avec une stratégie d'éclatement ou de fusion à la descente lors des ajouts.*

▸ **Exercice 3. Union-Find** (3 points)
Soit l'ensemble $S = \{0, 1, 2, 3, 4\}$.
En utilisant la fusion par rang et la compression des chemins, dessinez la forêt d'ensembles disjoints représentant les ensembles obtenus après chacune des opérations suivantes :

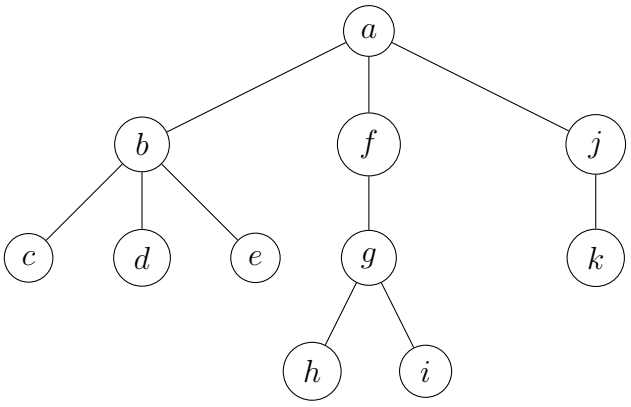
```
Union(0,1)
Union(1,3)
Union(4,2)
Union(3,2)
Union(2,4)
```

Pour chaque représentation, on précisera la valeur du rang de chaque arbre et on donnera sa représentation sous forme de tableau des pères.
Remarque : Lors d'une fusion, en cas d'égalité de rang, la nouvelle racine sera choisie comme le plus petit des représentants des éléments fusionnés.

▸ **Exercice 4. Représentation d'un arbre "fils gauche, frère droit"** (4 points)
On représente un arbre lexicographique à l'aide d'un arbre binaire "fils gauche, frère droit" La liste des frères est ordonnée.

```
typedef struct noeudL {
    char lettre;
    struct noeudL *fg,*frd;
} NoeudL, * ArbreL;
```

1. On considère l'arbre n-aire suivant :



   Donner sa représentation "fils gauche, frère droit".

2. Insérer les mots suivants dans un arbre lexicographique initialement vide.

   MAIS, ON, ILE, ONDE, MA, ILS, ILOT.

   Les arbres lexicographique seront représenté sous forme de "fils gauche, frère droit", les frères seront ordonnés par ordre alphabétique et le marqueur de fin de mot \0 sera conservé dans l'arbre.

   On rappelle que \0 *est inférieur à toutes les lettres de l'alphabet.*

3. Écrire une fonction `int nombreMots(ArbreL a)` qui renvoie le nombre de mots mémorisés dans l'arbre `a`.

4. Écrire une fonction `void affichePrefixe(ArbreL A, char *pref)` qui affiche les mots mémorisés par l'arbre `a` qui ont pour préfixe `pref`.

   Par exemple, avec l'arbre `A` construit à la question 2., l'appel `affichePrefixe(A,"m")` entraine l'affichage :

   ```
   MA
   MAIS
   ```

   alors que l'appel `affichePrefixe(A,"n")` entrainera l'affichage :

   ```
   Pas de mots trouvé !
   ```

▸ **Exercice 5. Tas** (2 points)

Dans cet exercice, on considère un tas binaire maximum (*i.e.* un nœud est plus grand que ses enfants).

Chacune des réponses aux questions suivantes devra être justifiée, i.e. *le résultat affirmé devra être démontré.*

1. Quelle est la complexité de l'ajout d'un élément dans un tas binaire ?
2. Quelle est la complexité de l'extraction de l'élément maximum du tas ?

▸ **Exercice 6. AVL** (1,5 points)

1. Quel est le nombre maximum de nœuds d'un arbre AVL de hauteur 6 ?
2. Quel est le nombre minimum de nœuds d'un arbre AVL de hauteur 6 ?

Justifiez vos réponses