EA4 – Éléments d'algorithmique II
TP n° 5 : arbres binaires de recherche

Le fichier `tp5.py` (importé dans `tp5_ex1.py`) fournit une interface permettant de créer et manipuler des arbres binaires :

- `Vide` : représente un arbre vide ;
- `Noeud(elt, arb_g, arb_d)` : crée un arbre dont l'étiquette de la racine est `elt`, le sous-arbre gauche est `arb_g` et le sous-arbre droit est `arb_d` ; aucun des arguments de la fonction ne doit être `None` ;
- `Feuille(elt)` : crée un arbre dont l'étiquette de la racine est `elt`, et les deux sous-arbres dont vides ;
- `estVide(arbre)` : renvoie `True` si et seulement si `arbre` est l'arbre vide ; l'argument `arbre` ne doit pas être `None` ;
- `etiquetteRacine(arbre)` : renvoie la valeur de l'étiquette de la racine de `arbre` ; l'argument `arbre` ne doit être ni `None` ni l'arbre vide ;
- `filsGauche(arbre)` : renvoie le sous-arbre gauche de `arbre` ; l'argument `arbre` ne doit être ni `None` ni l'arbre vide ;
- `filsDroit(arbre)` : renvoie le sous-arbre droit de `arbre` ; l'argument `arbre` ne doit être ni `None` ni l'arbre vide.

Les arbres devront être exclusivement manipulés via cette interface. En particulier, les nœuds ne sont pas modifiables, donc toute modification dans un sous-arbre nécessite la création (via un appel à `Noeud` ou `Feuille`) d'une nouvelle racine pour ce sous-arbre.

Ce fichier définit également une fonction `dessineArbreBinaire(arbre)` qui crée un fichier `arbre.pdf` représentant l'arbre passé en paramètre. Elle nécessite d'avoir installé `graphviz`[1] sur son ordinateur, qui permet (notamment) d'utiliser la commande `dot` dans le terminal.

Exercice 1 : recherches et ajouts dans un ABR

1. Écrire une fonction `parcoursInfixe(arbre)` qui renvoie la liste des étiquettes correspondant au parcours infixe de `arbre`.
2. Écrire une fonction `estUnABR(arbre)` qui renvoie `True` si les étiquettes des nœuds de l'`arbre` vérifient les conditions d'un ABR ***en temps linéaire en la taille de l'arbre***.
3. Écrire des fonctions `minimumABR(arbre)` et `maximumABR(arbre)` qui renvoient respectivement l'étiquette minimale et maximale dans `arbre`.
4. Écrire une fonction `rechercheABR(arbre, elt)` renvoyant `True` si et seulement si `elt` est contenu dans `arbre`.
5. Écrire une fonction `insertionABR(arbre, elt)` qui renvoie l'arbre correspondant à l'insertion de `elt` dans `arbre`. Si `elt` est déjà dans `arbre`, la fonction renvoie `arbre`.
6. Le test `testInsertion2` construit un ABR par insertions successives d'éléments à partir d'un arbre vide, puis crée les fichiers d'extension `.dot` et `.pdf` contenant un dessin de l'ABR. Sur le même modèle, vous pouvez construire d'autres ABR par ajouts successifs d'éléments. Représenter le résultat en utilisant la fonction `dessineArbreBinaire()`.

1. Pour l'obtenir avec votre gestionnaire de paquets : `apt-get install graphviz` (remplacer `apt-get` par `yum`, `brew`,... bref votre gestionnaire de paquets si ce n'est pas `apt`)

Exercice 2 : génération aléatoire par insertions successives

Dans cet exercice, on souhaite mesurer expérimentalement la hauteur moyenne d'un arbre construit par insertions successives à partir d'une permutation aléatoire de taille n.

1. Écrire (et tester) une fonction `hauteur(arbre)` qui renvoie la hauteur de l'`arbre`. On rappelle que la hauteur de l'arbre vide est -1.
2. Écrire une fonction `genererABRparInsertion(perm)` qui construit un arbre binaire de recherche par insertions successives des éléments de la permutation `perm`.
3. En utilisant les fonctions précédentes, ainsi que la fonction `permutation(n)` (qui renvoie une permutation aléatoire de taille `n`), écrire une fonction `statsHauteursABRparInsertion(n, m)` qui renvoie la liste des hauteurs de `m` arbres de taille `n` construits aléatoirement selon ce procédé.
4. À l'aide de la fonction `tracer(limite, pas, m)` fournie, observer la distribution des hauteurs des arbres construits de cette manière.

Exercice 3 : suppressions

1. Écrire la fonction `supprimerPlusPetit(arbre)` (resp. `supprimerPlusGrand(arbre)`) qui renvoie la paire `elt,new_arbre` où `elt` est le plus petit (resp. grand) élément de `arbre` et `new_arbre` est l'arbre `arbre` dans lequel `elt` a été supprimé. Renvoie la paire `None,Vide` quand `arbre` est vide.
2. Écrire (et tester) une fonction `suppressionABR(arbre, elt, alea=False)` qui supprime l'étiquette `elt` si elle existe selon la méthode vue en cours. Si `alea` vaut `False`, lorsque le nœud contenant l'élément à supprimer a 2 fils, on le remplacera par son ***prédécesseur***. Lorsque `alea` vaut `True` en revanche, on choisira à pile ou face le prédécesseur ou le successeur.
   Si `arbre` ne contient par l'étiquette `elt`, la fonction `suppressionABR(arbre, elt, alea=False)` devra renvoyer `None`.

Exercice 4 : génération aléatoire par insertions successives puis suppressions

Nous pouvons maintenant expérimenter des modèles un peu plus réalistes d'ABR aléatoires, obtenus à partir d'insertions mais également de suppressions.

1. Écrire une fonction `genererABRparInsPuisSup(perm)` qui construit un arbre de taille n selon le procédé suivant :
   - construire un arbre de taille n^2 par insertions successives à partir d'une permutation `perm` de taille n^2 ;
   - supprimer $n^2 - n$ éléments de l'arbre, chacun choisi uniformément parmi les éléments restants.
2. Écrire une fonction `statsHauteursABRparInsPuisSup(n, m)` qui produit la liste des hauteurs de `m` arbres de taille `n` obtenus par ce procédé à partir d'une permutation aléatoire, puis examiner les courbes obtenues.
3. Écrire une fonction `genererABRparInsEtSup(permins, permsup)` qui construit un arbre de taille au plus n à partir de deux permutations de taille n en alternant aléatoirement entre l'insertion d'un élément de `permins` et la suppression d'un élément de `permsup` (qui sera sans effet si l'élément n'est pas encore dans l'arbre). La fonction renverra un couple formé de l'arbre et de sa taille.
4. Écrire une fonction `statsHauteursABRparInsEtSup(n, m)` qui renvoie un tableau de couples constitués des tailles et hauteurs de `m` arbres obtenus par le procédé précédent à partir de deux permutations aléatoires de taille `2n`, puis examiner les courbes obtenues.