

EA4 – Éléments d’algorithmique II

TP n° 5 : arbres binaires de recherche (Correction)

Le fichier `arbre.py` fournit une interface permettant de créer et manipuler des arbres binaires :

- `Vide` : représente un arbre vide ;
- `Noeud(valeur, arb_g, arb_d)` : crée un arbre dont l’étiquette de la racine est `valeur`, 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(valeur)` : crée un arbre dont l’étiquette de la racine est `valeur`, et les deux sous-arbres sont 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 `.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*.

Les tests utilisés pour les questions 2 et 3 sont dessinés dans le dossier `./dessins_ex1/` (chaque fichier `arb_i.pdf` correspond au test numéro `i`).

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`.

Un des tests de la question suivante (intitulé `testInsertion2`) consiste à construire un arbre binaire de recherche grâce à des insertions successives. L’arbre résultant est dessiné dans le fichier `./dessins_ex1/monarbre.pdf`.

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. En vous inspirant de `testInsertion2`, 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. La fonction `tracer(limite, pas, m)` trace en fonction de n la hauteur de m arbres chacun obtenu par insertions successives de n éléments depuis une permutation aléatoire. Que pouvons-nous observer ?

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.

Les tests de la fonction du bloc TESTS sont dessinés automatiquement dans le dossier `./dessins_ex3/`. En cas d'échec du test `i`, le fichier `arbre_i` contient l'arbre de départ, `attenduPred_i` (resp. `attenduSuc_i`) contient le résultat attendu pour l'insertion avec prédécesseur (resp. successeur), et `obtenuPred_i` (resp. `obtenuSuc_i`) contient le résultat obtenu.

2. Écrire une fonction `suppressionABR(arbre, elt, pred=True)` qui supprime l'étiquette `elt` (si elle existe) selon la méthode vue en cours. Lorsque le nœud contenant l'élément à supprimer a deux fils, on le remplacera par son *prédécesseur* si `pred` vaut `True`, et par son *successeur* sinon. Si `arbre` ne contient pas l'étiquette `elt`, la fonction `suppressionABR(arbre, elt)` 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. La méthode de suppression (prédécesseur ou successeur) est choisie au hasard à chaque suppression.
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 méthode de suppression utilisée sera choisit au hasard à chaque suppression. 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.