

EA4 – Éléments d’algorithmique

TP n° 8 : arbres binaires de recherche

Les questions  seront à rendre sur Moodle avant dimanche 25 mars 16h (heure de Paris), sous forme d’une archive `tp8.tar` ou `tp8.zip`.





Attention : ce TP réutilise à peu près les notations du TP n° 7. On aura la même représentation des arbres binaires, par une liste de noeuds chacun de format :

[étiquette, filsGauche, filsDroit, père]

En revanche tous les arbres manipulés seront des arbres binaires «complétés», où chaque feuille est du type `[None, None, None, i]` où `i` est l’indice du père; il faudra s’assurer que cette propriété est préservée, en particulier lors de la modification d’un ABR.



Le fichier `tp8.py` (importé dans `tp8_ex1.py`) contient quelques fonctions utiles (`racine`, `copier`, `egalite`, `swap`, `suppr` ...)

Exercice 1 : recherches et ajouts dans un ABR

1.  Écrire une fonction `estUnABR(arbre)` qui renvoie `True` si les étiquettes des noeuds de l’`arbre` vérifient les conditions d’un ABR.
2.  Écrire des fonctions *non récursives* `minimumABR(arbre)` et `maximumABR(arbre)` qui retournent respectivement l’indice du noeud dont l’étiquette est la plus petite, et celui dont l’étiquette est la plus grande.
3.  Écrire une fonction `rechercheABR(arbre, elt)` retournant l’indice d’un noeud contenant `elt`, s’il en existe, et celui de la feuille vide à laquelle la recherche aboutit, sinon. Utiliser cette fonction pour écrire `contientABR(arbre, elt)` qui renvoie `True` si l’étiquette d’un des noeuds de l’`arbre` vaut `elt`, et `False` sinon.
4.  À l’aide de la fonction auxiliaire `rechercheABR(arbre, elt)` définie ci-dessus, écrire une fonction `insertionABR(arbre, elt)` qui ne fait rien si l’`arbre` contient `elt`, et l’insère à la bonne place dans l’`arbre` sinon.
5. Construire un ABR par ajouts successifs d’éléments. Représenter le résultat en utilisant la fonction `dessineArbreBinaire()`.

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 une fonction `genererABR(per)` qui construit un arbre binaire de recherche par insertions successives des éléments de la permutation `per`.
2.  En utilisant `genererABR(per)`, ainsi que les fonctions auxiliaires (fournies) `permutation(n)` (qui renvoie une permutation aléatoire de taille n) et `hauteurABR(arbre)` (qui renvoie la hauteur de l’`arbre`), écrire une fonction `statABR(n, m)` qui renvoie la hauteur moyenne de m arbres de taille n construits aléatoirement selon le procédé.
3. Quelle est l’ordre de grandeur de la hauteur moyenne ?

Exercice 3 : suppressions

1. ✎ Écrire une fonction `maximum2ABR(arbre, i)` qui retourne l'indice du nœud d'étiquette maximale dans le sous-arbre enraciné en le nœud d'indice i .
2. ✎ Écrire une fonction `relier(arbre, ipere, ifils, d)` où `ipere` et `ifils` sont deux indices de nœud, et `d` un entier qui vaut soit 1 soit 2 :
 - si `d` vaut 1, la fonction doit relier le nœud d'indice `ifils` comme fils gauche du nœud d'indice `ipere`
 - si `d` vaut 2, la fonction doit relier le nœud d'indice `ifils` comme fils droit du nœud d'indice `ipere`
3. ✎ Écrire une fonction `suppressionABR(arbre, elt)` qui supprime le nœud d'étiquette `elt` s'il existe.

On supprimera un élément du tableau avec la fonction `suppr` fournie dans `tp8.py`, qui prend en arguments un arbre et un tableau d'indices et supprime ces nœuds de l'arbre. **Attention**, cette fonction ne perturbe pas l'arbre mais permute des éléments dans la représentation de l'arbre. Il faut donc l'appeler à la toute fin de la fonction et une seule fois.

On prendra les conventions suivantes :

- cas 0 : si l'étiquette ne correspond à aucun nœud, on ne fait rien.
- cas 1 : si le nœud à supprimer est une étiquette terminale (ses deux fils sont des feuilles vides), il suffit de remplacer ce nœud par une feuille vide.
- cas 2 : si le nœud n'a qu'un fils non vide (l'autre étant réduit à une feuille vide), il suffit de faire pointer directement le père de ce nœud vers le fils non vide.
- cas 3 : si le nœud (notons le p) a deux fils non vides, alors soit q le prédécesseur de p . On remplace l'étiquette de p par celle de q puis on supprime q (qui n'a pas de fils droit) dans le sous-arbre gauche de p .

Exercice 4 : génération aléatoire par insertions successives puis suppressions

On souhaite maintenant regarder un modèle un peu plus réaliste d'ABR aléatoire, obtenu à partir d'insertions mais également de suppressions. Pour construire un arbre aléatoire de taille n , on utilise le procédé suivant :

- construire un arbre de taille n^2 par insertions successives à partir d'une permutation aléatoire de taille n^2 ;
 - supprimer $n^2 - n$ éléments de l'arbre, chacun choisi uniformément parmi les éléments restants.
1. Écrire une fonction `generer2ABR(per)` qui construit, à partir de la permutation `per` de taille n^2 , un arbre binaire de recherche de taille n , en appliquant le procédé décrit ci-dessus.
 2. Écrire alors une fonction `stat2ABR(n, m)` qui renvoie la hauteur moyenne de m arbres de taille n construits aléatoirement selon ce nouveau procédé.
 3. Comparer les hauteurs moyennes obtenues par ce procédé et celui de l'exercice 2.