Binary Search Trees : BST

1 Parcours

Un *arbre binaire de recherche* (*binary search tree*) est un arbre binaire étiqueté tel que pour tout nœud v de l'arbre :

— les éléments de tous les nœuds du sous-arbre gauche de v sont inférieurs ou égaux à l'élément contenu dans v,

— et les éléments de tous les nœuds du sous-arbre droit de v sont supérieurs à l'élément contenu dans v.

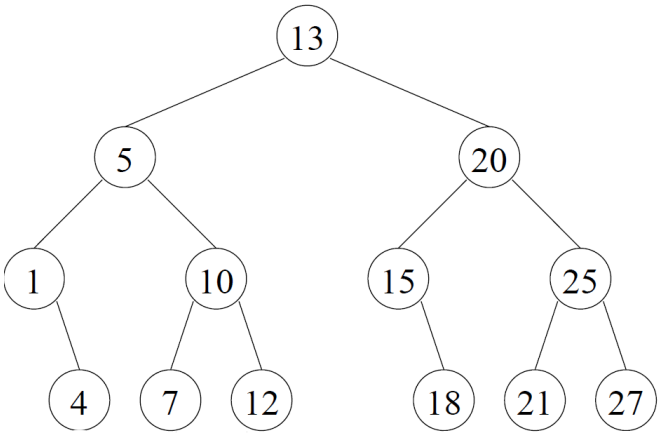

FIGURE 1 – BST

Exercice 1 (Test)

Écrire une fonction qui vérifie si un arbre binaire est bien un arbre binaire de recherche.

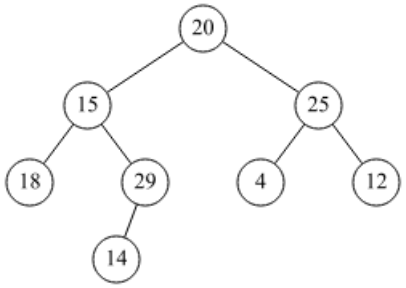

FIGURE 2 – ABR ?

Exercice 2 (Recherches)

1. (a) Où se trouvent les valeurs minimum et maximum dans un arbre binaire de recherche non vide ?
   (b) Écrire les deux fonctions `minBST(`B`)` (en récursif) et `maxBST(`B`)` (en itératif), avec B un ABR non vide.
2. Écrire une fonction qui recherche une valeur x dans un arbre binaire de recherche. La fonction retournera l'arbre contenant x en racine si la recherche est positive, la valeur `None` sinon.
   Deux versions pour cette fonction : récursive et itérative !

---

2 Modifications

Exercice 3 (Insertion en feuille)

1. En utilisant le principe de l'ajout aux feuilles, construisez, à partir d'un arbre vide, l'arbre binaire de recherche obtenu après ajouts successifs des valeurs suivantes (dans cet ordre) :
   $$13, 20, 5, 1, 15, 10, 18, 25, 4, 21, 27, 7, 12$$
2. Écrire une fonction récursive qui ajoute un élément dans un arbre binaire de recherche.
   Bonus : écrire la fonction d'insertion en itératif.

Exercice 4 (Suppression)

Écrire une fonction récursive qui supprime un élément dans un arbre binaire de recherche.

---

3 Bonus

Exercice 5 (Insertion en racine)

1. En utilisant le principe de l'ajout en racine, construisez, à partir d'un arbre vide, l'arbre binaire de recherche obtenu après ajouts successifs des valeurs suivantes (dans cet ordre) :
   $$13, 20, 5, 1, 15, 10, 18, 25, 4, 21, 27, 7, 12$$
2. Écrire la fonction qui ajoute un élément en racine dans un *arbre binaire de recherche*.