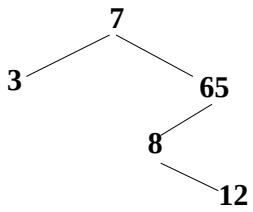
Arbre binaire d'entiers en dynamique

1 Definitions

Un arbre binaire est une structure qui est soit vide, soit composée par une racine et deux sous-arbres : gauche et droit. Il est défini d'une façon récursive :

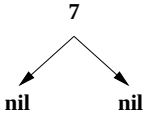
Arbre = Vide | <racine, gauche, droit>
racine = Elément
gauche, droit = Arbre

Exemple :

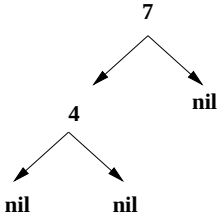


Dans cet arbre, 7 est la racine, le sous-arbre ayant 3 pour racine est le sous-arbre gauche alors que le sous-arbre ayant 65 pour racine est le sous-arbre droit.
Un arbre binaire peut servir pour structurer les données dans le but de les traiter d'une façon mieux optimisée. Dans l'exemple ci-dessus si on parcourt l'arbre avec la politique GRD (gauche-racine-droit), on obtient les valeurs triées : 3-7-8-12-65.

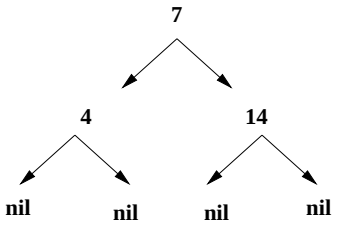
Construction d'un arbre GRD : La politique GRD peut se décrire de la manière suivante : on range chaque nombre d'une suite de nombres en utilisant le premier nombre de la suite comme racine pour l'arbre, puis les autres nombres récursivement en respectant la contrainte suivante : ceux qui sont inférieurs ou égaux sont placés dans le sous-arbre gauche, les autres sont placés dans le sous-arbre droit.
Si on décompose la construction de l'arbre pour la suite de nombres (7, 4, 14, 2, 6, 12, 15, 0, 5, 3), on obtient :
– étape 1, on traite le premier nombre 7 ; c'est la racine (**nil** représente le sous-arbre vide)



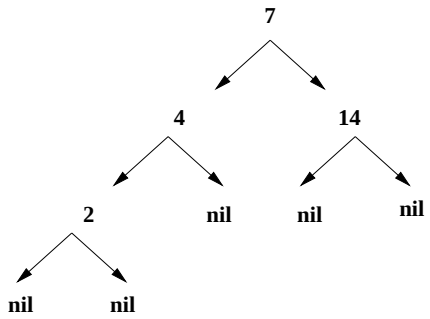
– étape 2, on traite le nombre suivant 4 ; il est < ou = à 7, on le met dans le sous-arbre gauche de 7



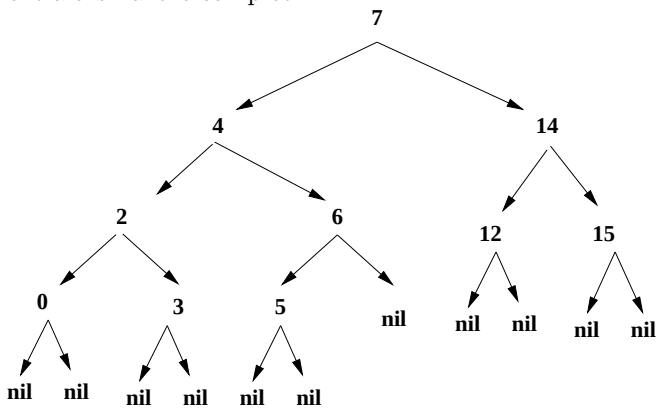
– étape 3, on traite le nombre suivant 14 ; il est > à 7, on le met dans le sous-arbre droit de 7



– étape 4, on traite le nombre suivant 2 ; il est $<$ ou $=$ à 4, on le met dans le sous-arbre gauche de 4



– et ainsi de suite ; on obtient alors l'arbre complet



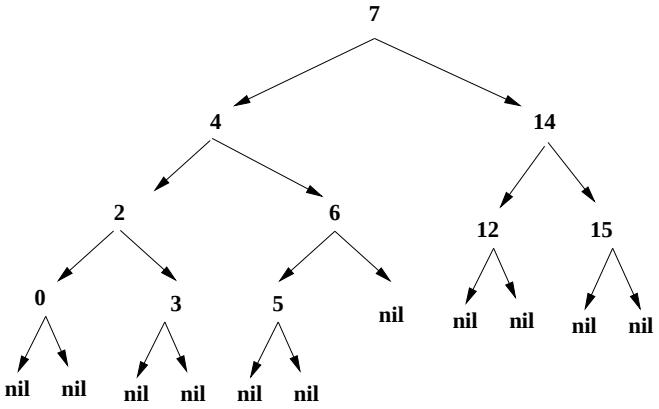
Le TAD correspondant est :

Sorte : ARBB
Utilise : int
Constructeurs :
créer : $\rightarrow$ ARBB
construire : int X ARBB X ARBB $\rightarrow$ ARBB
Projecteurs :
racine : ARBB $\rightarrow$ int
gauche : ARBB $\rightarrow$ ARBB
droit : ARBB $\rightarrow$ ARBB
estvide : ARBB $\rightarrow$ boolean
Préconditions
a :ARBB
racine(a), gauche(a), droit(a) défssi not estvide(a)
Axiomes
racine(construire(r,ag,ad)) = r
gauche(construire(r,ag,ad)) = ag
droit(construire(r,ag,ad)) = ad
estvide(créer) = true
estvide(construire(r,ag,ad)) = false

Remarquons que ce TAD peut aussi se définir à partir d'un autre constructeur qui ajoute juste un nouvel entier à l'arbre en respectant la méthode GRD :

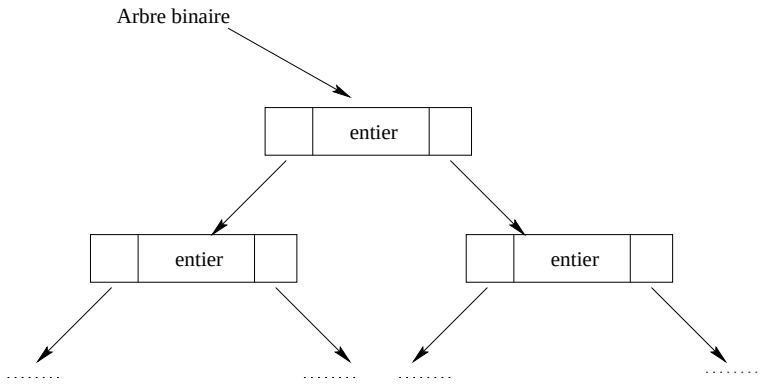
ajout : int X ARBB $\rightarrow$ ARBB

Les parcours Pour être efficaces, ils dépendent de l'implémentation. Sur l'arbre suivant, le parcours en profondeur donnera la suite 0-2-3-4-5-6-7-12-14-15, alors que le parcours en largeur donnera la suite 7-4-14-2-6-12-15-0-3-5 :



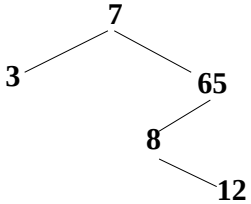
2 Implémentation en dynamique

Un arbre binaire d'entiers est représenté par un pointeur vers une cellule qui contient d'une part un entier et d'autre part deux pointeurs vers d'autres cellules.



Créer l'unité Arbre binaire et le fichier de test associé qui permettent de :

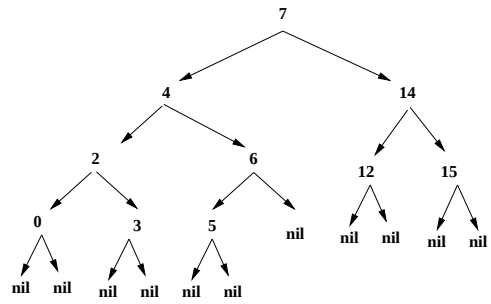
1. Définir le type ARBRE d'entier(dynamique).
2. Définir le sous-programme INITIALISE l'arbre binaire (donné en paramètre) .
3. Définir un sous-programme qui permet de tester si l'arbre (donné en paramètre) est VIDE.
4. Définir un sous-programme qui permet d'ajouter un entier (donné en paramètre) à un arbre (donnée en paramètre). Pour ajouter un entier, on doit parcourir l'arbre de façon à positionner l'entier à sa place : l'arbre est trié en GRD (gauche racine droite : le plus petit entier est le plus à gauche possible de l'arbre).
   Exemple : insérer successivement 7, 65, 8, 12 et 3 donne l'arbre suivant :



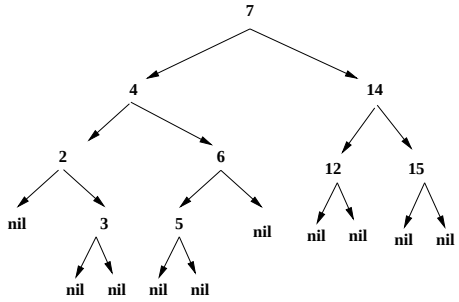
5. Définir un sous-programme qui permet d'enlever le plus petit élément de l'arbre (donné en paramètre) ce sous-programme doit renvoyer l'élément en question.
6. Définir un sous-programme récursif AFFICHE qui permet d'afficher tous les éléments de l'arbre dans l'ordre croissant.

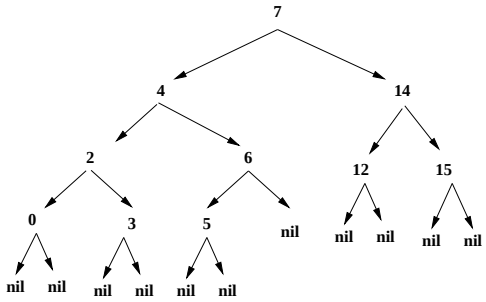
Rq : Cela donnera très facilement un parcours en profondeur.

7. écrire la fonction qui permet de supprimer un entier d'un arbre GRD, par exemple :

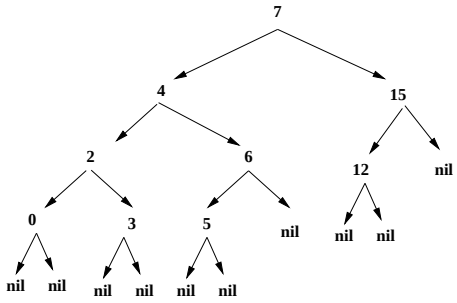


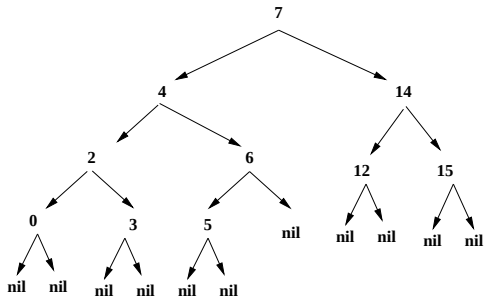
Si on supprime 0 :



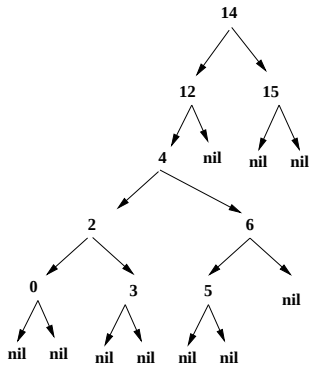


Si on supprime 12 :





Si on supprime 7 :



8. (a) Peut-on transformer le sous-programme récursif AFFICHE en un sous-programme itératif, qui fait la même chose, sans utiliser de structure de données auxilliaire et sans modifier le type ARBRE ?

   (b) Si non, quelle structure de données auxilliaire, autre qu'un arbre proposez-vous ? Donnez alors le sous-programme itératif correspondant.

   (c) Peut-on éviter la structure de données auxilliaire en modifiant le type ARBRE ? Donnez alors le nouveau type ARBRE et le sous-programme itératif correspondant.

   Rq : La réponse à cette question permet de trouver comment réaliser un parcours en largeur pour un arbre défini en dynamique.