Aix*Marseille université

Domaine Sciences et Technologies
Portail 1 René Descartes

Introduction à l'informatique : TD 9

Code UE : S14IN1I1

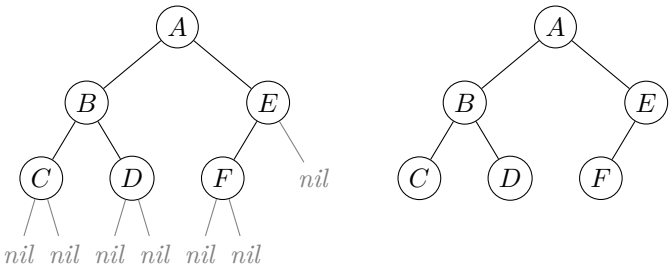
Année 2020–21

Arbres binaires

Un arbre est un graphe non orienté connexe (c'est-à-dire *d'un seul morceau*), sans cycle. On a vu en cours que, dans la plupart des applications, on distingue un nœud [1] particulier d'un arbre, qu'on appelle la racine : on dessine alors l'arbre avec la racine en haut (ils sont fous ces informaticiens !). Une restriction consiste alors à considérer des arbres *binaires*, dont on a vu une définition récursive. Un arbre binaire est :

— soit l'arbre vide, qu'on note souvent *nil* (et qu'on ne représente généralement pas dans les dessins) ;
— soit une racine ayant un enfant gauche et un enfant droit, qui sont tous deux des arbres binaires.

Ainsi, à gauche ci-dessous le graphe est un arbre binaire, qu'on représente dans la suite comme dessiné à droite, sans les arbres *nil* :



La racine de cet arbre est le nœud A qui a deux enfants : l'enfant gauche a B pour racine, et l'enfant droit a E pour racine. Les enfants du nœud C sont deux arbres vides *nil*. Une feuille est un nœud qui possède l'arbre vide comme enfants gauche et droit : les feuilles de l'arbre du dessus sont C, D et F.

Exercice 1 On connaît trois algorithmes de parcours d'arbre : le parcours préfixe, postfixe (ou suffixe) et infixe. Ces trois parcours ne diffèrent que par l'ordre dans lequel on visite la racine, l'enfant gauche et l'enfant droit :

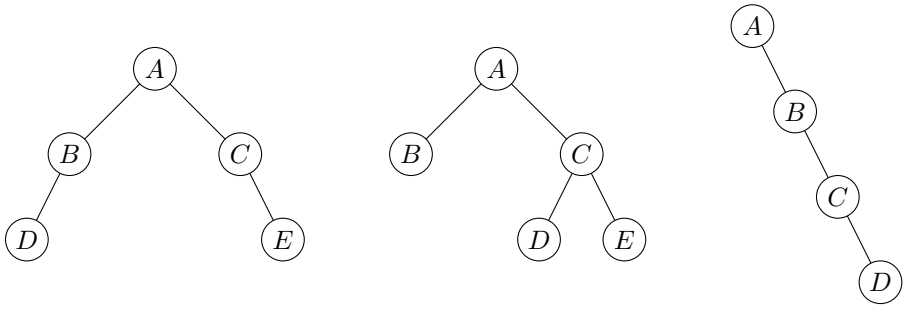
```
fonction parcours_préfixe(nœud) :
    Si nœud ≠ nil alors
        afficher(valeur[nœud])
        parcours_préfixe(enfant_gauche[nœud])
        parcours_préfixe(enfant_droit[nœud])

fonction parcours_postfixe(nœud) :
    Si nœud ≠ nil alors
        parcours_postfixe(enfant_gauche[nœud])
        parcours_postfixe(enfant_droit[nœud])
        afficher(valeur[nœud])

fonction parcours_infixe(nœud) :
    Si nœud ≠ nil alors
        parcours_infixe(enfant_gauche[nœud])
        afficher(valeur[nœud])
        parcours_infixe(enfant_droit[nœud])
```

1. Nœud et sommet sont des synonymes dans ce cours.

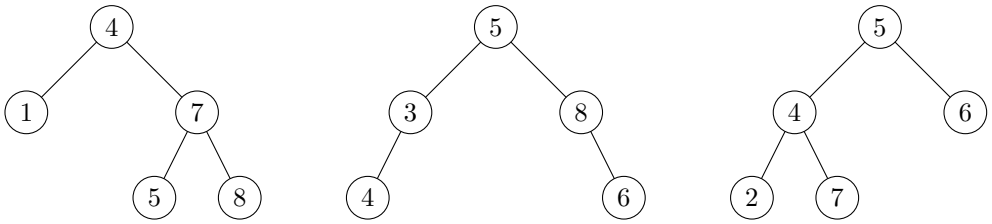
1. Appliquer les trois algorithmes à la racine des arbres binaires suivants, en montrant les valeurs affichées.



2. Trouver un arbre binaire dont le parcours préfixe est A, B, C, D, E, F et le parcours infixe est C, B, E, D, F, A. (*Attention, on cherche bien un seul arbre qui admet ces deux parcours à la fois !*)
3. Trouver deux arbres binaires différents dont le parcours préfixe est A, B, D, C, E, G, F et le parcours postfixe est D, B, G, E, F, C, A.

Exercice 2 Un *arbre binaire de recherche* (ABR) est un arbre tel que chaque nœud x de l'arbre vérifie la propriété suivante : toutes les valeurs des nœuds dans l'enfant gauche de x sont inférieures à la valeur de x, et toutes les valeurs des nœuds dans l'enfant droit de x sont supérieures à la valeur de x.

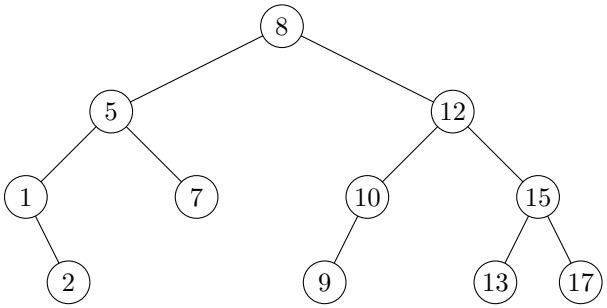
1. Lesquels des arbres suivants sont des ABR et pourquoi ?



2. Un arbre binaire de recherche sert à stocker un ensemble de valeurs (un dictionnaire ou un annuaire par exemple). Une opération importante est donc la recherche d'une valeur particulière dans cet ensemble. On a vu en cours un algorithme récursif permettant d'effectuer cette recherche :

```
fonction rechercher_abr(nœud, x)
    si nœud = nil alors
        retourner Faux
    sinon si x = valeur[nœud] alors
        retourner Vrai
    sinon si x < valeur[nœud] alors
        retourner rechercher_abr(enfant_gauche[nœud], x)
    sinon
        retourner rechercher_abr(enfant_droit[nœud], x)
```

Appliquer cet algorithme pour rechercher les valeurs 15, 7, 2, 11, 3 dans l'ABR suivant, en donnant la liste des nœuds où l'algorithme s'appelle récursivement.



3. Une autre opération intéressante pour un ABR est la possibilité d'insérer un élément nouveau à l'intérieur : c'est crucial si on veut représenter un annuaire qu'on peut mettre à jour facilement. Construire un ABR en insérant une par une (à partir de l'arbre vide) les valeurs $9, 5, 12, 2, 15, 7, 11$ dans l'ordre. Ensuite, construire un autre ABR en insérant les mêmes valeurs, mais dans l'ordre $2, 5, 7, 9, 11, 12, 15$. Quelle est la hauteur des deux arbres [2] et quelle différence y aura-t-il en termes de complexité lors d'une recherche dans le pire des cas ?
4. Appliquer l'algorithme de parcours infixe aux arbres binaires construits dans la question précédente. Dans quel ordre l'algorithme affiche-t-il les valeurs ?
5. En déduire un algorithme de tri de tableau (qu'on pourra décrire avec des phrases) qui utilise un ABR comme structure de données auxiliaire.

2. La hauteur d'un arbre est la longueur de la plus grande branche qui relie la racine à une feuille de l'arbre.