LU2IN003
Arbres binaires (suite) - Exercices de base semaine 7

Alix Munier et Maryse Pelletier

Version : mars 2020

Exercice(s)

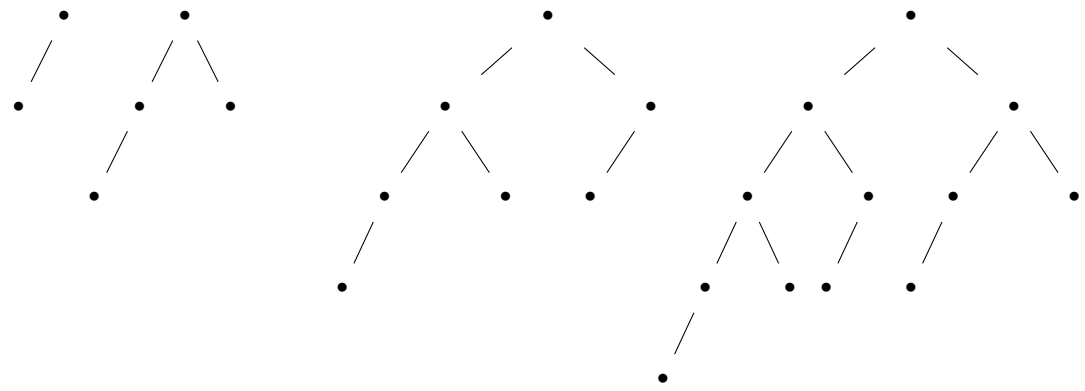
1 Arbres H-équilibrés

Exercice 1 – Arbres binaires et nombres de Fibonacci

On rappelle la définition des nombres de Fibonacci : $F_0 = 0$, $F_1 = 1$ et $F_k = F_{k-1} + F_{k-2}$ si $k \geq 2$.
On donne la définition inductive suivante des *arbres de Fibonacci* :

— $T_0 = \emptyset$, $T_1 = \bullet$
— $T_k = (\bullet, T_{k-1}, T_{k-2})$ si $k \geq 2$

Question 1
Dessiner T_2, T_3, T_4, T_5.
Solution:



Question 2
Montrer que $n(T_k) = F_{k+2} - 1$, pour tout $k \in \mathbb{N}$ par induction structurelle.
Solution:

Base Pour T_0, $n(T_0) = 0$ et $F_2 - 1 = 0$. Pour T_1, $n(T_1) = 1$ et $F_3 - 1 = 1$. La propriété est donc vraie pour $k = 0$ et pour $k = 1$.

Induction Pour $k \geq 2$, soit $T_k = (\bullet, T_{k-1}, T_{k-2})$ avec $n(T_{k-1}) = F_{k+1} - 1$ et $n(T_{k-2}) = F_k - 1$. On a alors

$$n(T_k) = n(T_{k-1}) + n(T_{k-2}) + 1 = F_{k+1} - 1 + F_k - 1 + 1 = F_{k+2} - 1$$

et la propriété est vraie pour T_k.

Conclusion La propriété est vérifiée par induction structurelle.

Question 3

Montrez par induction structurelle que, pour tout $k \geq 0$, $h(T_k) = k$ et que T_k est H-équilibré.

Solution:

Base Pour $k = 0$, $h(T_0) = 0$ et pour $k = 1$, $h(T_1) = 1$. De plus, T_0 et T_1 sont H-équilibrés.

Induction Pour $k \geq 2$, soit $T_k = (\bullet, T_{k-1}, T_{k-2})$ avec $h(T_{k-1}) = k-1$ et $n(T_{k-2}) = k-2$. De plus, on suppose que T_{k-1} et T_{k-2} sont H-équilibrés. On a alors $h(T_k) = 1+\max(h(T_{k-1}, h(T_{k-2}) = 1+\max(k-1, k-2) = k$.

De plus, $h(T_{k-1}) - h(T_{k-2}) = 1$. On en déduit que T_k est H-équilibré.

Conclusion La propriété est vérifiée par induction structurelle.

2 Arbres parfaits et tas

Exercice 2 – Hauteur et taille d'un arbre parfait

Question 1

Montrer que la taille n d'un arbre parfait de hauteur h vérifie $2^{h-1} \leq n < 2^h$.

Solution:

Soit T un arbre parfait de hauteur h et de taille n alors $n \leq 2^h - 1$ d'après l'exercice 1 du TD 6. Donc $n < 2^h$. Les $h-1$ premiers niveaux de T sont entièrement remplis et contiennent donc $\sum_{i=1}^{h-1} 2^{i-1} = \sum_{i=0}^{h-2} 2^i = 2^{h-1} - 1$ nœuds. il y a au moins un nœud au niveau h donc $2^{h-1} \leq n$.

Question 2

En déduire la valeur de h en fonction de n. Quelle est la hauteur d'un arbre parfait de 10000 sommets ?

Solution:

En appliquant le logarithme, on obtient $h - 1 \leq \log_2(n) < h$ et donc $h = 1 + \lceil n \rceil$. $\log_2(10000) = 13,28$, donc $h = 14$.

Exercice 3 – Opérations sur les tas

Dans cet exercice on considère des arbres binaires étiquetés par des nombres.

Un *tournoi* est un arbre binaire dont les étiquettes croissent de la racine vers les feuilles. Un *tas* est un tournoi parfait. Le *parcours par niveau* d'un arbre parfait est la liste obtenue en parcourant les nœuds de l'arbre niveau par niveau et de gauche à droite.

Question 1

Pour chacun des trois arbres binaires T_1, T_2, T_3 suivants, dire s'il est parfait, s'il est un tas. Justifier les réponses.

Solution:

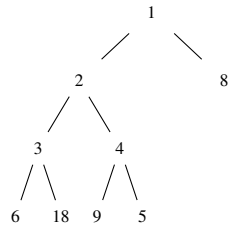
T_1 n'est pas parfait, car l'avant-dernier niveau n'est pas entièrement rempli.

T_2 est un tas : il est parfait et les étiquettes croissent de la racine vers les feuilles.

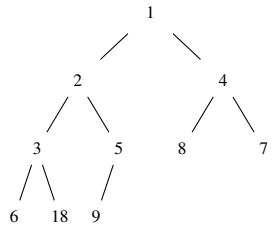
T_3 est parfait mais n'est pas un tas : sur le chemin de la racine vers la feuille d'étiquette 2, on a $5 > 2$.

On rappelle qu'un arbre parfait de taille n peut être représenté au moyen d'un tableau $A[0..N]$, avec $N \geq n$, tel que :

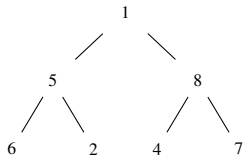
Arbre T_1 :



Arbre T_2 :

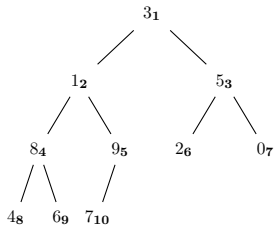


Arbre T_3 :



— $A[0]$ contient la taille n de T
— les cases $A[1..n]$ sont remplies en parcourant T de gauche à droite, niveau par niveau.

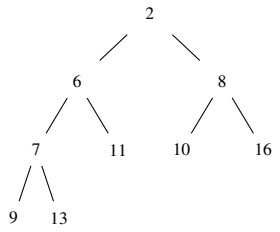
Ce tableau A est appelé *parcours par niveau* de l'arbre parfait T.
On peut numéroter les nœuds de l'arbre parfait T au moyen de leur position dans le tableau A comme illustré dans l'exemple suivant :



Le tableau associé à est $[10, 3, 1, 5, 8, 4, 9, 2, 0, 4, 6, 7]$

Question 2

On considère le tas ExT :

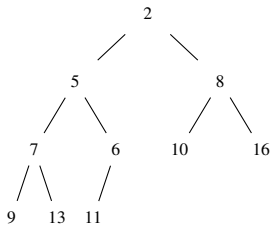


1. Donner le tableau ExA associé à ExT.
2. Réaliser l'insertion de la clé 5 dans le tas ExT, en maintenant la structure de tas. Décrire brièvement (deux phrases maximum) l'algorithme utilisé. Donner le tableau associé au résultat.
3. Réaliser la suppression de la clé minimale dans le tas ExT, en maintenant la structure de tas. Décrire brièvement (deux phrases maximum) l'algorithme utilisé. Donner le tableau associé au résultat.

Solution:

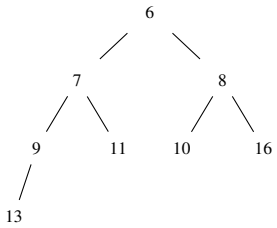
1. $ExA = [9, 2, 6, 8, 7, 11, 10, 16, 9, 13]$

2. On insère la nouvelle valeur v à la fin du tableau et on la fait remonter à sa bonne place dans le tas. Pour cela, on échange v avec la clé de son père tant que v est inférieure à la clé de son père. Tableau :



$[10, 2, 5, 8, 7, 6, 10, 16, 9, 13, 11]$.

3. On remplace la clé de la racine par la valeur v du dernier élément du tableau et on fait redescendre v à sa bonne place. Pour cela, on échange v avec le plus petit de ses fils tant que v est supérieure à au moins l'un de ses fils. Tableau : $[8, 6, 7, 8, 9, 11, 10, 16, 13]$.



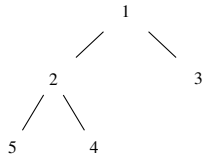
Question 3

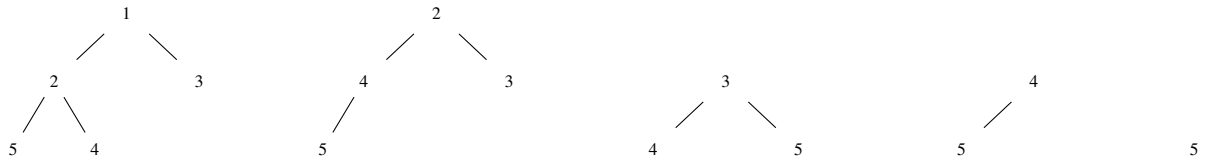
Soit L une liste de n entiers naturels.

1. Décrire brièvement (deux phrases maximum) l'algorithme permettant de trier la liste L en utilisant un tas. Quelle en est la complexité en nombre de comparaisons dans le pire cas ?
2. Soit $L_0 = (5, 2, 3, 1, 4)$.
   (a) Construire le tas T_0 obtenu en insérant successivement les éléments de L_0 et le tableau A_0 associé à ce tas.
   (b) Détruire le tas T_0 par suppressions successives du minimum. On dessinera l'arbre obtenu à chaque suppression et on donnera le tableau associé à chaque tas.

Solution:

1. On construit le tas obtenu en insérant successivement les éléments de L puis on détruit le tas par suppressions successives du minimum. La complexité pire cas est en $O(n \log n)$.
2. $L_0 = (5, 2, 3, 1, 4)$.
   (a) Tas T_0 :
   Taleau $A_0 = [5, 1, 2, 3, 5, 4]$
   (b) Valeurs successives du tas :
   Valeurs successives du tableau : $[5, 1, 2, 3, 5, 4]$, $[4, 2, 4, 3, 5, 1]$, $[3, 3, 4, 5, 2, 1]$, $[2, 4, 5, 3, 2, 1]$, $[1, 5, 4, 3, 2, 1]$ et enfin $[0, 5, 4, 3, 2, 1]$.



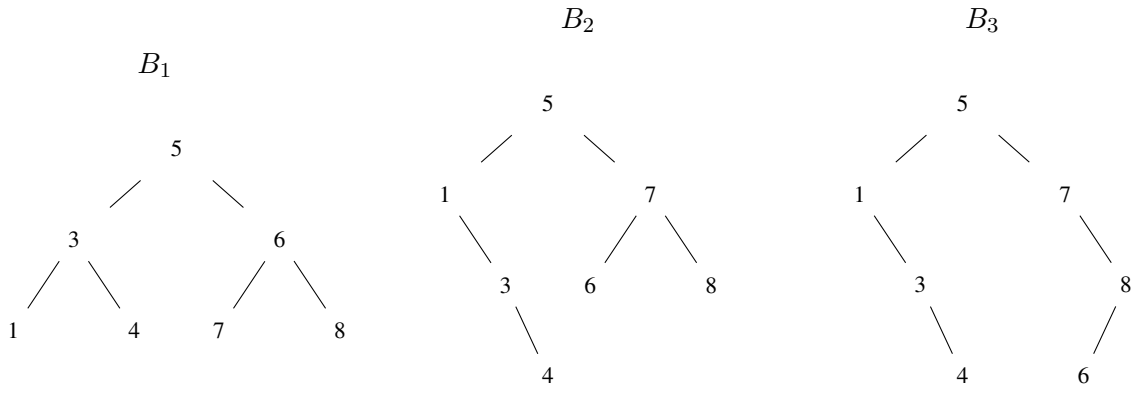


3 Arbres binaires de recherche (ABR)

Exercice 4 – Définition des ABR

Question 1

Est-ce que les arbres B_1, B_2 et B_3 sont des ABR ? Justifier les réponses négatives.



Solution:
B_1 n'est pas un ABR car la clef 7 est supérieure à la clef 6. B_2 est un ABR. B_3 n'est pas un ABR car la clef 6 est inférieure à la clef 7.

Exercice 5 – Caractérisation d'un ABR

On rappelle que, si T est un ABR, `ABinfixe(T)` est une liste rangée en ordre strictement croissant.

Question 1

1. Dessiner trois ABR différents dont les clefs sont $3, 1, 4, 8, 5, 2, 7, 6$. Donner le parcours infixe de chacun d'eux.
2. Pouvez-vous dessiner deux ABR différents ayant la même forme et les mêmes clefs (deux à deux distinctes) ?

Solution:

1. Il suffit de dessiner trois ABR dont le parcours infixe est $1, 2, 3, 4, 5, 6, 7, 8$.
2. Soit T_1 et T_2 deux ABR ayant la même forme et les mêmes clefs deux à deux distinctes. Alors T_1 et T_2 ont le même parcours infixe : la liste des clefs rangée en ordre croissant. Par hypothèse T_1 et T_2 ont la même forme, c'est-à-dire la même ombre. On peut donc conclure que T_1 et T_2 sont égaux (exercice 4 du TD 6).

Question 2

Montrer que, si le parcours infixe des clefs d'un arbre binaire T est rangé en ordre strictement croissant alors T est un ABR. Cette propriété peut être démontrée par récurrence sur la taille de l'arbre.

Solution:

On démontre cette propriété par récurrence forte sur la taille de l'arbre.

Base La propriété est vérifiée pour un arbre de taille 0.

Induction Soit $n > 0$, supposons que la propriété soit vraie pour tout arbre de taille $k < n$. Soit $T = (c, G, D)$ un arbre binaire de taille n tel que le parcours infixe des clefs de T soit rangé en ordre strictement croissant. Ainsi, tous les éléments de son fils gauche sont strictement plus petits que c, tous ceux de son fils droit sont strictement plus grands. Par récurrence forte,

— G est un arbre de taille $k < n$ dont le parcours infixe des clefs est rangé en ordre strictement croissant donc G est un ABR,

— D est un arbre de taille $k' < n$ dont le parcours infixe des clefs est rangé en ordre strictement croissant donc D est un ABR.

Par conséquent T est un ABR.

Question 3

1. Décrire en une phrase le principe d'un algorithme qui teste si un arbre binaire est un ABR.
2. En supposant que les listes sont représentées par des listes circulaires doublement chaînées, calculer la complexité de cet algorithme.

Solution:

1. On calcule le parcours infixe de l'arbre T et on teste si la liste obtenue est strictement croissante. On obtient les codes Python qui suivent.

   Parcours infixe :

```
def ABinfixe(T):
  if estABvide(T):
    return []
  else:
    return ABinfixe(T.gauche) + [T.clef] + ABinfixe(T.droit)
```

   Test de croissance d'une liste :

```
def estCroissante(L):
  if len(L) == 0 or len(L) == 1:
    return True
  return (L[0] < L[1]) and estCroissante(L[1:])
```

   Test ABR :

```
def estABR1(T):
  return estCroissante(ABinfixe(T))
```

2. Chaque concaténation de listes est en $\Theta(1)$ puisque les listes sont représentées par des listes circulaires doublement chaînées. La complexité de `ABinfixe` est donc en $\Theta(n)$ où n est la taille de l'arbre.

   La complexité de `estCroissante` est en $\mathcal{O}(n)$ (pire cas) et en $\Omega(1)$ (meilleur cas), où n est la taille de la liste.

   La complexité de `estABR1` est en $\Theta(n)$ où n est la taille de l'arbre.

Question 4

Soit P une liste dont les éléments sont deux à deux distincts. Montrer que, s'il existe un ABR dont le parcours préfixe est P, alors cet arbre est unique.

Solution:

Le parcours infixe de l'ABR est la liste I obtenue en rangeant la liste P en ordre croissant. D'après l'exercice 3, question 2 de la feuille de TD 6 sur les arbres binaires, il existe un unique arbre binaire dont le parcours préfixe est P et le parcours infixe I.

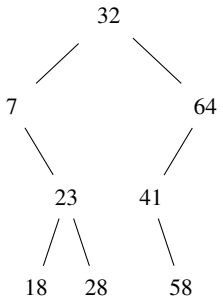
Exercice 6 – Exemple de manipulation d'un ABR

Question 1

L'ajout de nouvelles clefs dans un ABR se fait par insertion aux feuilles. Construire l'ABR Tex obtenu par insertion successive des clefs 32, 7, 23,64, 18, 28, 41 et 58. Est-ce que cet ABR est unique ? Que se passe-t'il si l'on change l'ordre des clefs ?

Solution:

Voici l'arbre obtenu par insertion successive des clefs 32, 7, 23,64, 18, 28, 41 et 58 :



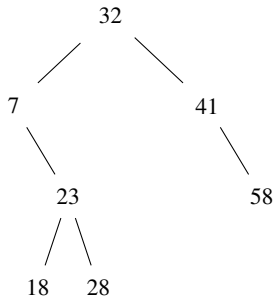
Cet ABR est unique pour un ordre d'insertion des clefs fixé. Si on change l'ordre des clefs lors de l'insertion, alors on obtient un autre ABR.

Question 2

Donner l'ABR obtenu en supprimant le maximum de Tex.

Solution:

On obtient l'arbre suivant :



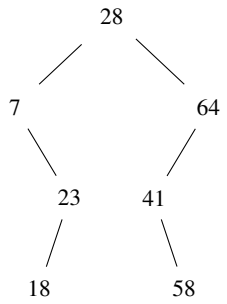
(On supprime l'élément le plus grand et on remonte son fils gauche qui prend sa place.)

Question 3

Donner l'ABR obtenu en supprimant la racine de Tex.

Solution:

On obtient l'arbre suivant :



La racine a un fils gauche : on la remplace la racine par le plus grand élément de son fils gauche (et cet élément est supprimé).

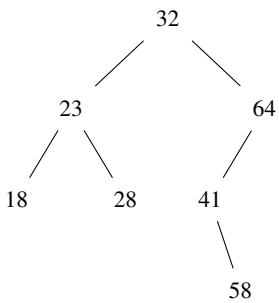
Question 4

Donner l'ABR obtenu en supprimant la clef 7 dans Tex.

Solution:

Cela revient à supprimer la racine du sous-arbre dont la racine a pour clef 7 : il suffit de remplacer le nœud de clef 7 par son fils droit.

On obtient l'arbre suivant :



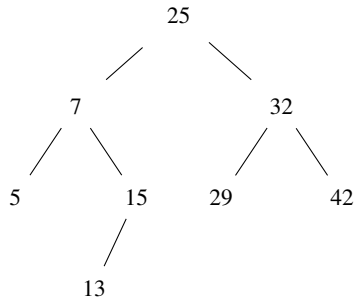
Exercice 7 – Insertion d'une clef dans un ABR

Dans cet exercice, on étudie l'algorithme d'insertion d'une clef dans un ABR rappelé à la suite :

```
def ABRinsertion(x,T):
  if estABvide(T):
    return ABfeuille(x)
  if x == T.clef:
    return T
  if x < T.clef:
    return AB(T.clef,ABRinsertion(x,T.gauche),T.droit)
  return AB(T.clef,T.gauche,ABRinsertion(x,T.droit))
```

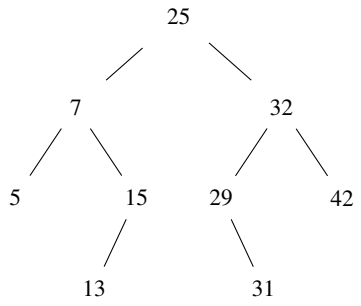
Question 1

On considère dans cette question l'ABR suivant. Donner son parcours infixe. Conclusion ? Construire l'ABR obtenu après l'insertion de la clef 31. Donnez alors le parcours infixe de ce nouvel ABR.



Solution:

Le parcours infixe de cet arbre est $I = (5, 7, 13, 15, 25, 29, 32, 42)$. Comment la liste I est strictement croissante, on en déduit que l'arbre donné est bien un ABR. Après l'insertion de la clef 31, on obtient l'ABR suivant :



Le parcours infixe de ce nouvel arbre est $I = (5, 7, 13, 15, 25, 29, 31, 32, 42)$. C'est donc bien un ABR.

Question 2

Montrer que `ABRinsertion(x,T)` se termine et retourne un ABR dont les clés sont x et les clés de T.

Solution:

Notons $\mathcal{P}(T)$ la propriété : pour tout x, `ABRinsertion(x,T)` se termine et retourne un ABR dont les clés sont x et les clés de T.

Montrons $\mathcal{P}(T)$ par induction.

Base $\mathcal{P}(\emptyset)$ est vérifiée : terminaison immédiate et renvoi de l'arbre réduit à la feuille x.

induction Soit $T = (c, G, D)$ un ABR, supposons que $\mathcal{P}(G)$ et $\mathcal{P}(D)$ soient vraies.

— Si $x = c$ alors `ABRinsertion(x,T)` se termine immédiatement et retourne T qui est un ABR dont les clés sont x et les clés de T, puisque x est une clé de T.

— Si $x < c$ alors `ABRinsertion(x,T)` fait un appel à `ABRinsertion(x,G)` qui, par hypothèse, se termine et retourne un ABR G' dont les clés sont x et les clés de G. Par conséquent `ABRinsertion(x,T)` se termine et retourne un arbre binaire $T' = (c, G', D)$, qui est un ABR et dont les clés sont x et les clés de T.

— Le cas $x > c$ est analogue au précédent.

Conclusion La propriété $\mathcal{P}(T)$ est vérifiée pour tout ABR T.