Algo 2 – séance 5
Les arbres binaires de recherche (ABR)

Denis Trystram

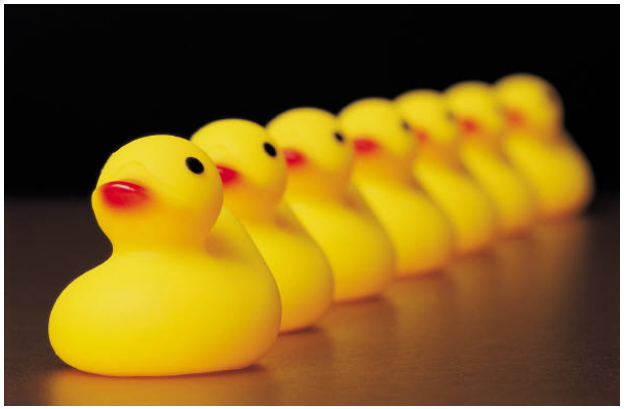
6 mars 2018

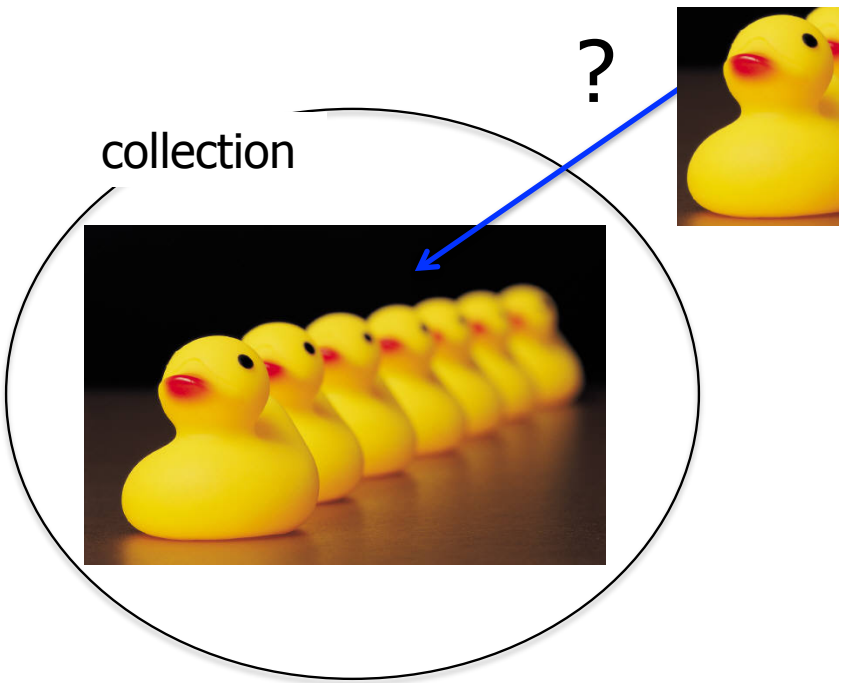
Structures de données avancées, dictionnaires

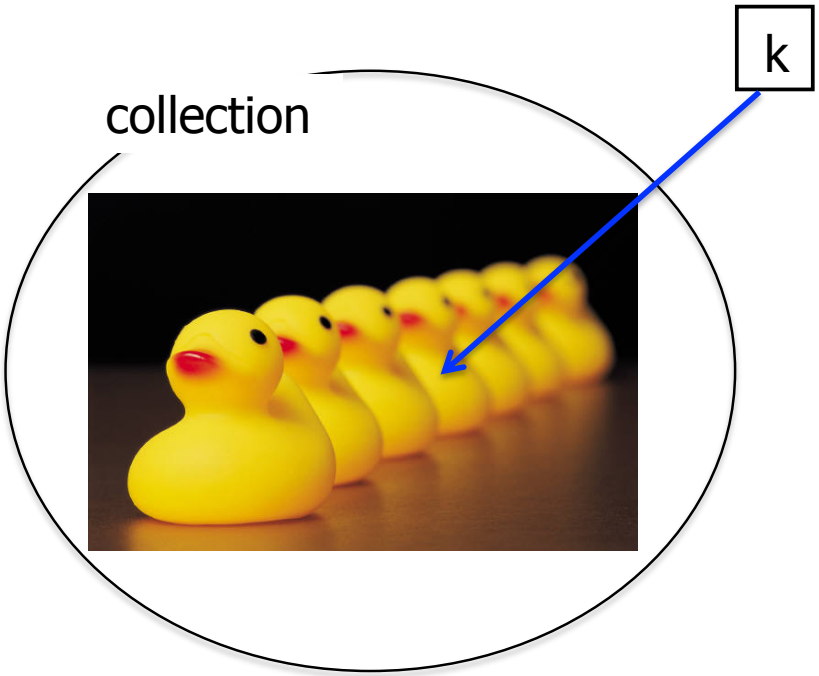
Les opérations usuelles sont les suivantes :

- recherche d'un élément
- insertion
- suppression

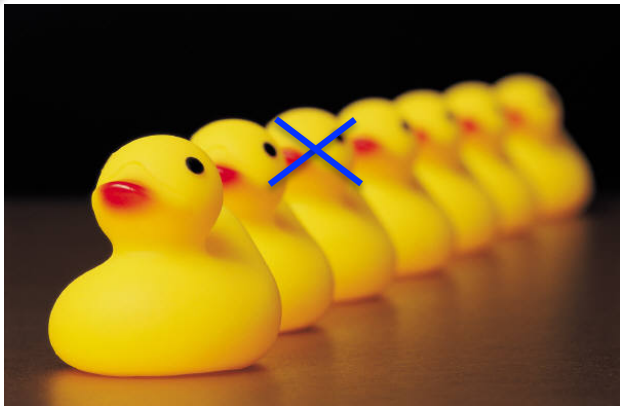
collection




?
collection


k
collection

collection



Plan de la séance

- Définition, préliminaires
- Quelques opérations fondamentales
  - Parcours (simple)
  - Recherche d'un élément
  - Recherche d'un élément extrémal
  - Successeur
  - Insertion d'un nouvel élément (fait en TD)
- Complexité (en moyenne)

Outline

Motivation

Les Arbres Binaires de Recherche (ABR) sont des structures de données qui offrent des coûts moyens en $(log_2 n)$ pour toutes les opérations classiques d'un dictionnaire (recherche, insertion ou suppression d'un élément).

La variante des ABR équilibrés garantie un coût en $(log_2 n)$ aussi en pire cas.

Motivation

Les Arbres Binaires de Recherche (ABR) sont des structures de données qui offrent des coûts moyens en $(log_2 n)$ pour toutes les opérations classiques d'un dictionnaire (recherche, insertion ou suppression d'un élément).

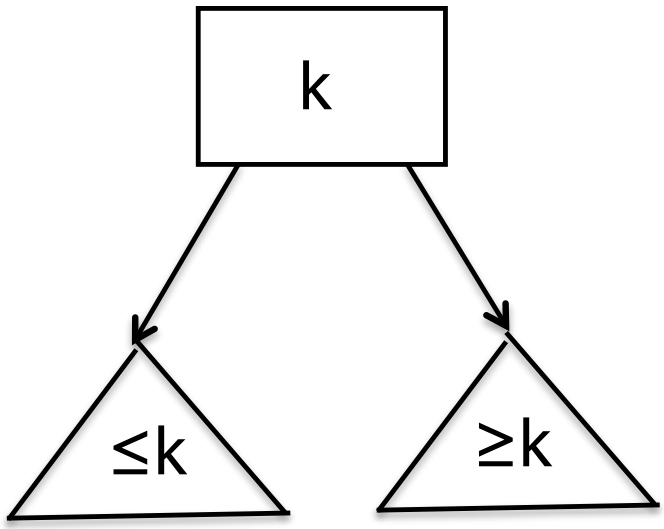
La variante des ABR équilibrés garantie un coût en $(log_2 n)$ aussi en pire cas.

Définition

Les ABR sont des structures de données en arbres binaires dont les sommets sont accédés par des *clés* (des entiers) et qui ont la propriété invariante suivante :

Pour tout sommet x, toutes les clés du sous-arbre gauche sont inférieures à x et toutes celles du sous-arbre droit lui sont supérieures.

Schéma de principe



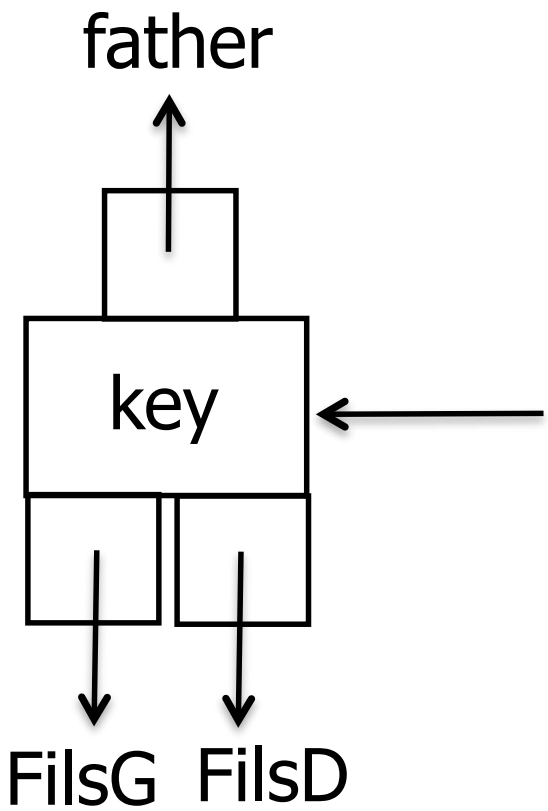
Formellement

Structure élémentaire :

```
type binaryTree is access Node
type Node is record
    key : integer
    FilsG, FilsD : binaryTree
end record
```

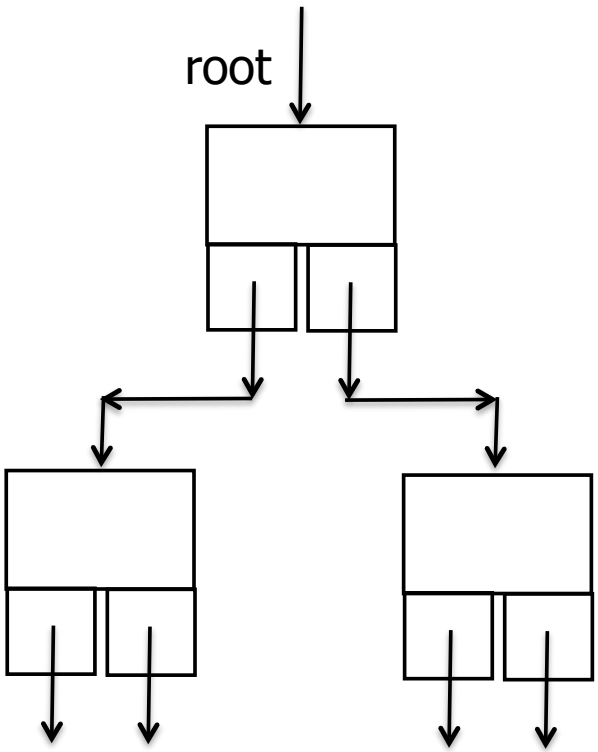
On peut ajouter un champ *Father*, souvent utile.

Brique de base: type Node



Structure d'un ABR

On assemble les briques de base comme dans un mécano...



Principe général

La structure ABR est la base qui peut être adaptée selon les opérations visées.

Le principe est de rajouter un *invariant* pour garantir une propriété supplémentaire qui conduira à un gain selon la cible.
Ces propriétés d'invariance sont (évidemment) plus compliquée à mettre en œuvre ou/et à étudier...[1]

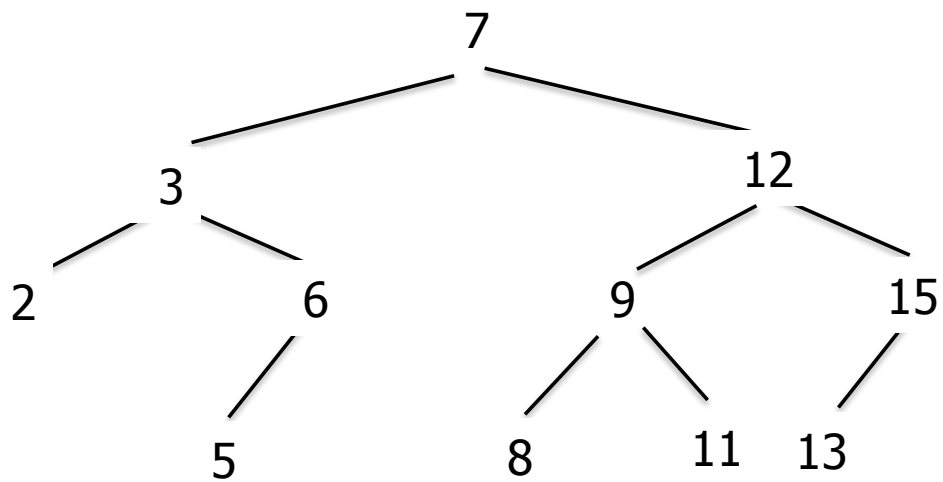
[1] no free lunch!

Exemple

Soit la séquence : 7, 12, 9, 3, 11, 6, 2, 15, 5, 8, 13, ...

Exemple

7, 12, 9, 3, 11, 6, 2, 15, 5, 8, 13, ...



Généralisation

On peut remplacer *FilsG* et *FilsD* par un tableau de binaryTree (*children[i]*) à indices booléens.

Par exemple, A.children[0] = false et A.children[1]= true.

Avantage : découpler la structure du parcours.
Simplicité d'écriture de structures plus complexes comme les *Btrees* (sujet d'examen de juin 2016). On remplace alors les booléens par une fonction.

Quelques manipulations de base sur les ABR

Sans perte de généralités, on suppose que tous les éléments d'un ABR sont différents et que les éléments que l'on cherche sont bien présents dans l'ABR considéré.

On cherche tout d'abord un algorithme récursif simple pour afficher toutes les clés triées d'un ABR.

Des idées ?

Solution : Affichage dans l'ordre des clés

Algorithm 1: parcoursABR

Input: un ABR A

```
1 if A! = null then
2 |   parcoursABR(A.FilsG);
3 |   print (A.key);
4 |   parcoursABR(A.FilsD);
5 end
```

Cette solution (qui n'est autre qu'un **parcours infixé**) est obtenue par appel à "parcoursABR(T.root)" pour l'ABR T.
On montre facilement que le coût de cet algorithme est linéaire en nombre de sommets.

Solution : Affichage dans l'ordre des clés

Algorithm 2: parcoursABR

Input: un ABR A

```
1 if A! = null then
2 |   parcoursABR(A.FilsG);
3 |   print (A.key);
4 |   parcoursABR(A.FilsD);
5 end
```

Cette solution (qui n'est autre qu'un **parcours infixé**) est obtenue par appel à "parcoursABR(T.root)" pour l'ABR T.
On montre facilement que le coût de cet algorithme est linéaire en nombre de sommets.

Recherche d'un élément

Une autre opération simple et facile à écrire est la recherche d'un élément à une clé donnée.

Principe

On part de la racine et on descend en comparant les clés.

Ecrire la version récursive.

Recherche d'un élément

Une autre opération simple et facile à écrire est la recherche d'un élément à une clé donnée.

Principe

On part de la racine et on descend en comparant les clés.

Ecrire la version récursive.

Version récursive

Algorithm 3: SearchABR

Input: un ABR A et un entier k

```
1 if A.key == k then
2 |   return A
3 end
4 if k < A.key then
5 |   SearchABR(A.FilsG,k)
6 end
7 else
8 |   SearchABR(A.FilsD,k)
9 end
```

Comme la récursivité est terminale, on peut réécrire l'algorithme en itératif (souvent plus efficace).

Coût

C'est juste la longueur d'un chemin de la racine à une racine (au pire):
en $\mathcal{O}(h)$ où h est la hauteur de l'arbre.

Si l'arbre est dégénéré (réduit à une chaine), c'est linéaire.
Si c'est un arbre complet, c'est $log_2(n)$.

Une autre structure

On remplace *FilsG* et *FilsD* par un tableau d'arbres *children[i]* dont l'indice 0 correspond à *FilsG* (et 1 à *FilsD*).
Manipulation plus facile et généralisable à d'autres types d'arbres.
Exemple sur la recherche *search(noeud,k)* :

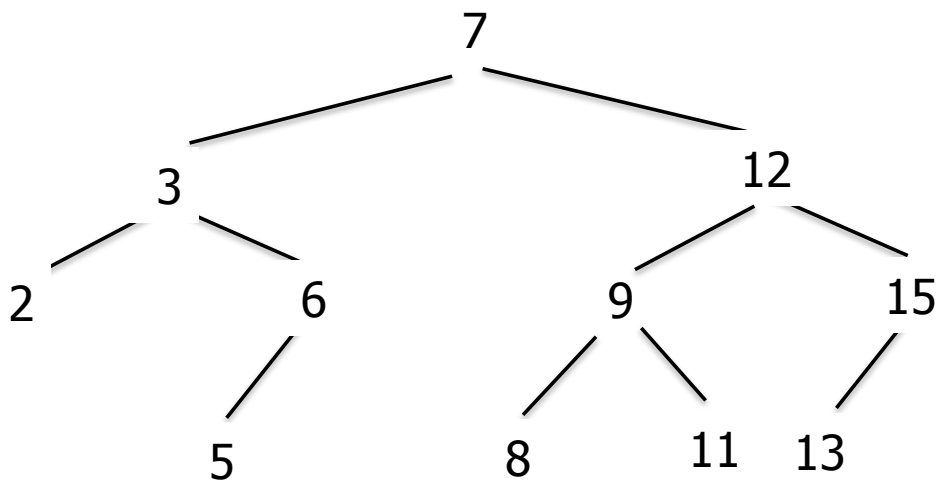
Algorithm 4: search

Input: un ABR A, un entier k

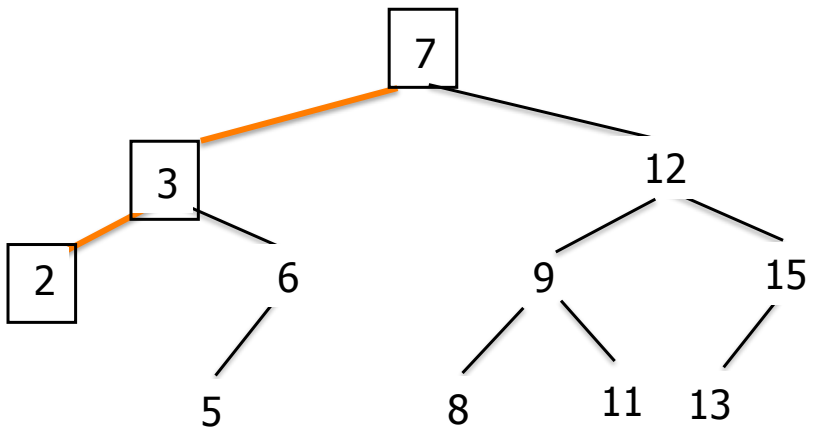
```
1 if A.key == k then
2 |   return A;
3 end
4 else
5 |   return search(A.children[k>A.key],k));
6 end
```

Variante : déterminer l'élément minimal ou maximal d'un ABR

Comment caractériser le min ?



C'est l'élément le plus à gauche possible !



Déterminer l'élément minimal

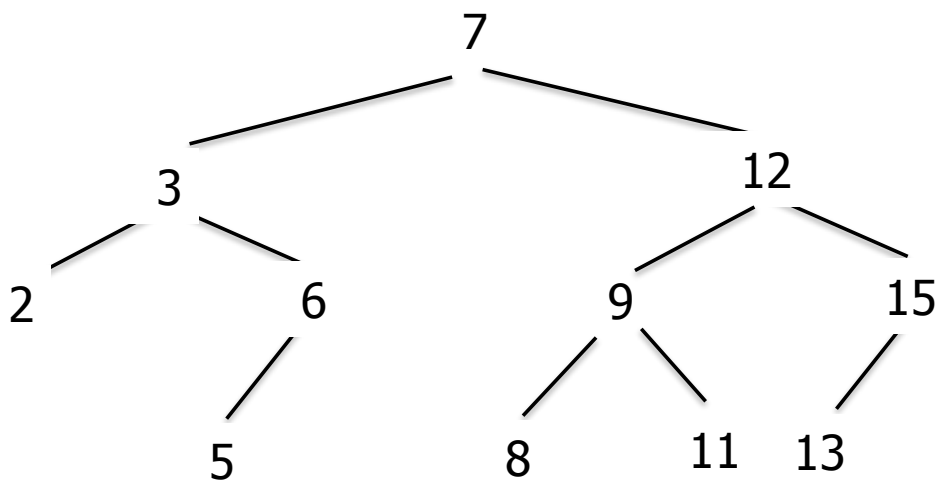
Algorithm 5: findMin

Input: un ABR A

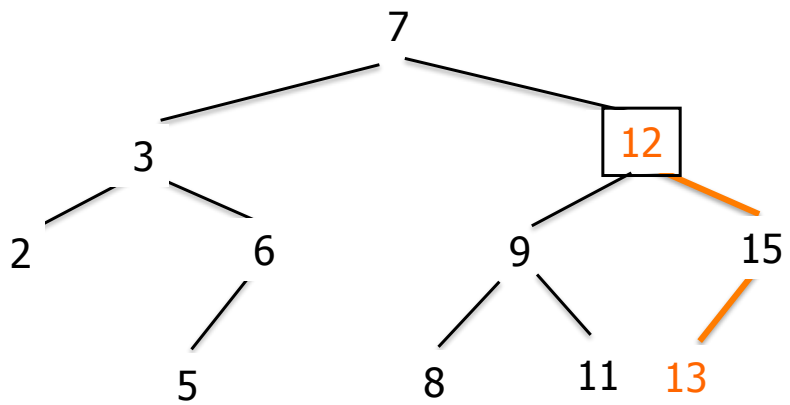
```
1 while A.FilsG != null do
2 |   A ← A.FilsG
3 end
4 return A
```

Successeur d'un élément

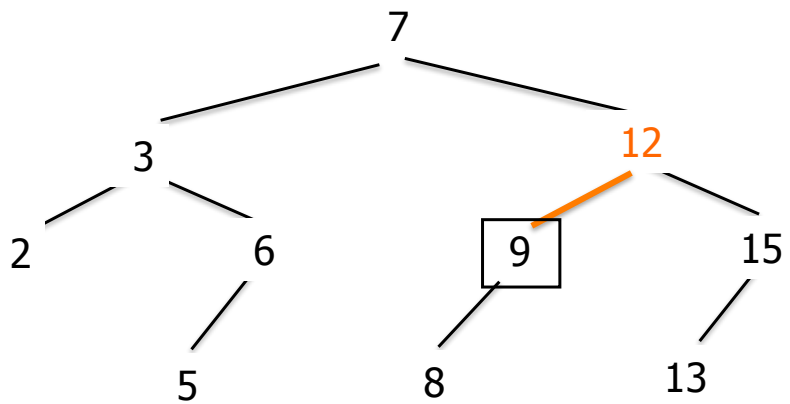
Le successeur d'un sommet est défini comme l'élément suivant dans l'ordre de la séquence.



Cas 1. FindMin(15)



Cas 2. Father(9)



Successeur

La fonction *Succ(x)* utilise l'algorithme précédent *findMin*.
La structure même d'ABR permet d'effectuer cette opération sans faire de comparaisons entre les clés !

Etant donné un sommet x, on doit distinguer deux cas selon que x possède ou non un fils droit.

- S'il possède un fils droit, il suffit alors d'appeler *findMin(x.FilsD)*.
- Sinon, la propriété d'invariance garantie que le successeur de x est le premier ancêtre de x dont le fils de gauche est aussi un ancêtre de x. En d'autres termes, le sommet cherché est le père du sous-arbre gauche maximal.

Successeur (algorithme)

Algorithm 6: Succ

Input: un sommet x dans un ABR

```
1 if x.FilsD != null then
2 |   return findMin(x.FilsD)
3 end
4 y ← x.Father;
5 while (y != null) and (y.FilsD = x) do
6 |   x ← y;
7 |   y ← x.Father
8 end
9 return y
```

Définition des B-Trees

L'idée ici est de créer une structure plus efficaces que les ABR de base qui mêle les structures d'arbre et de liste.
Chaque noeud contient plusieurs clés du dictionnaire.

Définition d'un B-Tree de dégré d ($d \geq 2$).

k : nombre de clés stockées dans le noeud x.
Une liste des clés (ordre croissant).
Un booléen *EstUneFeuille*.
Un pointeur pour chaque noeud fils de x.

Le nombre de fils de x est exactement $k+1$ (sauf si x est une feuille).
$d \leq k+1 \leq 2d$ (sauf la racine qui peut en avoir moins de d.
Les feuilles sont toutes situées à la même hauteur de l'arbre.

Définition des B-Trees

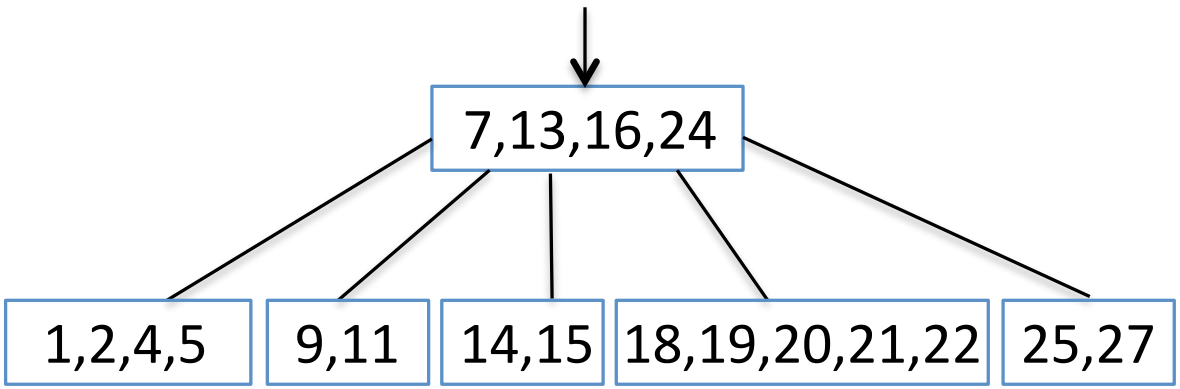
L'idée ici est de créer une structure plus efficaces que les ABR de base qui mêle les structures d'arbre et de liste.
Chaque noeud contient plusieurs clés du dictionnaire.

Définition d'un B-Tree de dégré d ($d \geq 2$).

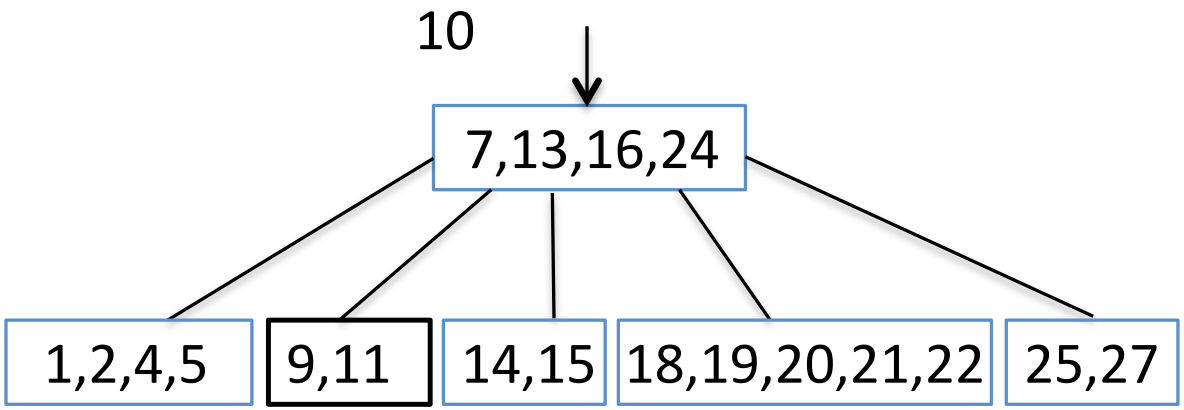
k : nombre de clés stockées dans le noeud x.
Une liste des clés (ordre croissant).
Un booléen *EstUneFeuille*.
Un pointeur pour chaque noeud fils de x.

Le nombre de fils de x est exactement $k+1$ (sauf si x est une feuille).
$d \leq k+1 \leq 2d$ (sauf la racine qui peut en avoir moins de d.
Les feuilles sont toutes situées à la même hauteur de l'arbre.

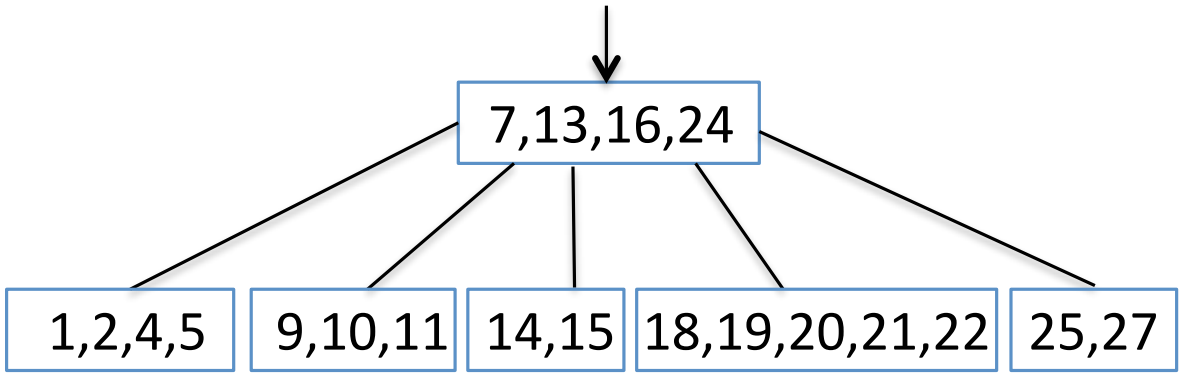
Exemple d'insertion (d=3)



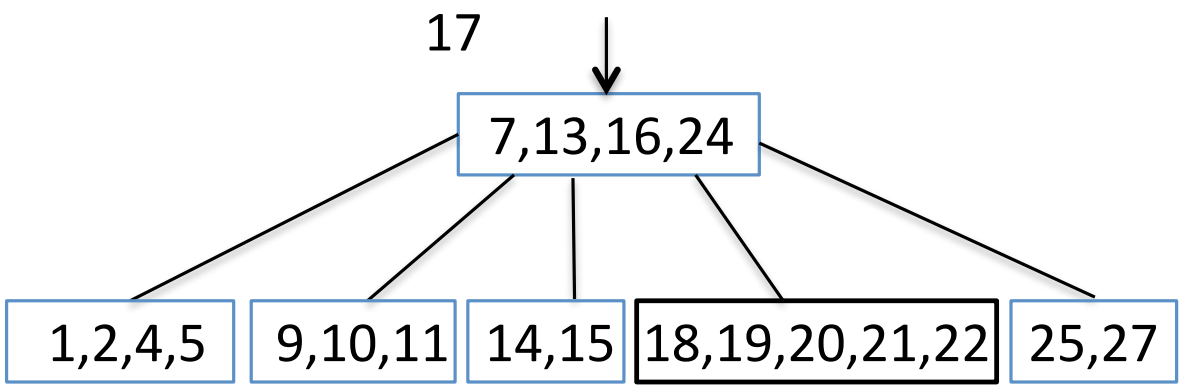
Exemple - insertion de 10



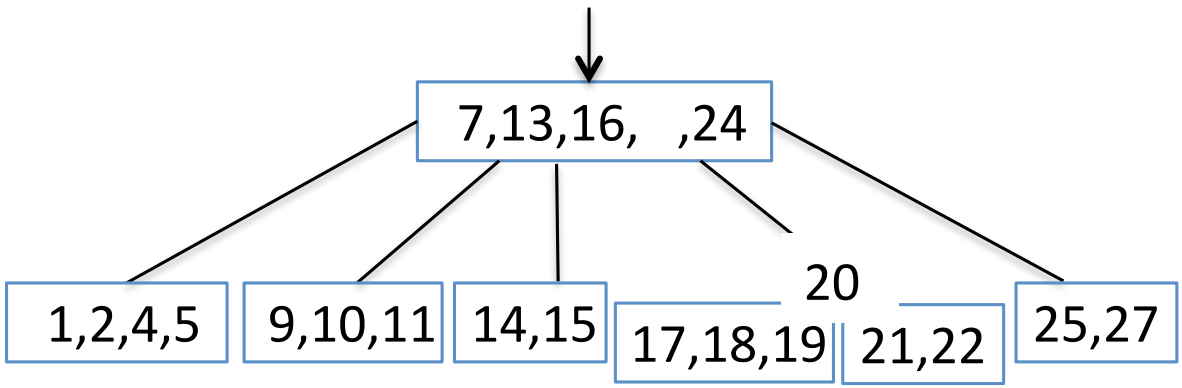
Exemple - insertion de 10 (final)



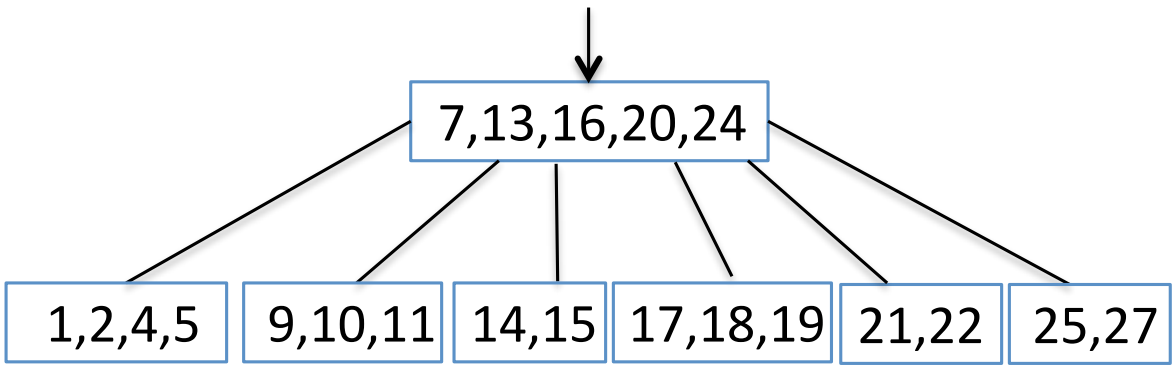
Exemple - insertion de 17 (noeud saturé)



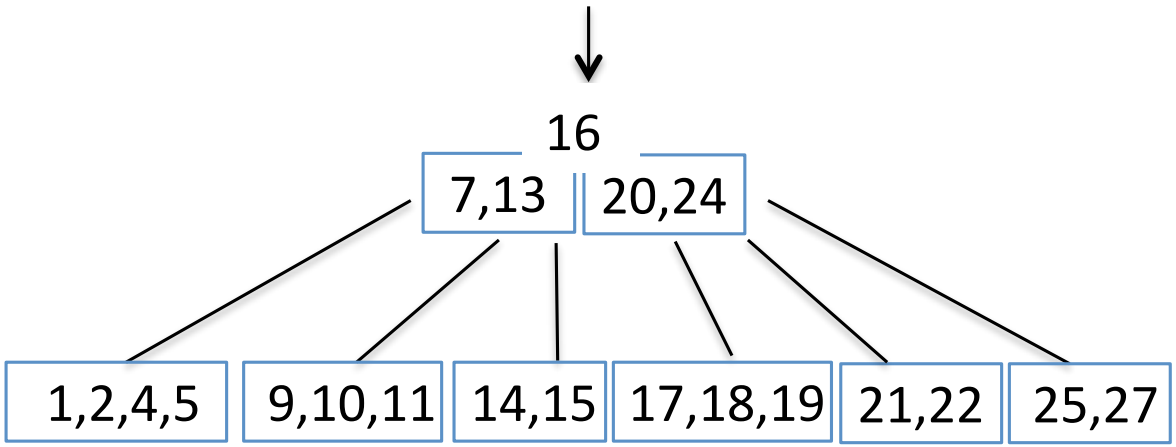
Exemple - recomposition du sous-arbre



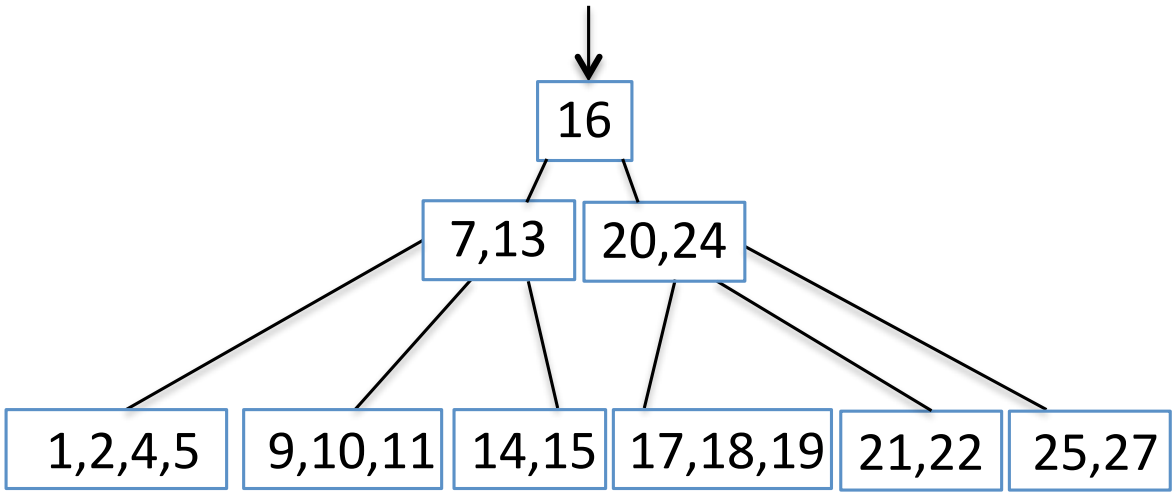
Exemple - 20 remonte à la racine



Exemple - on éclate la racine

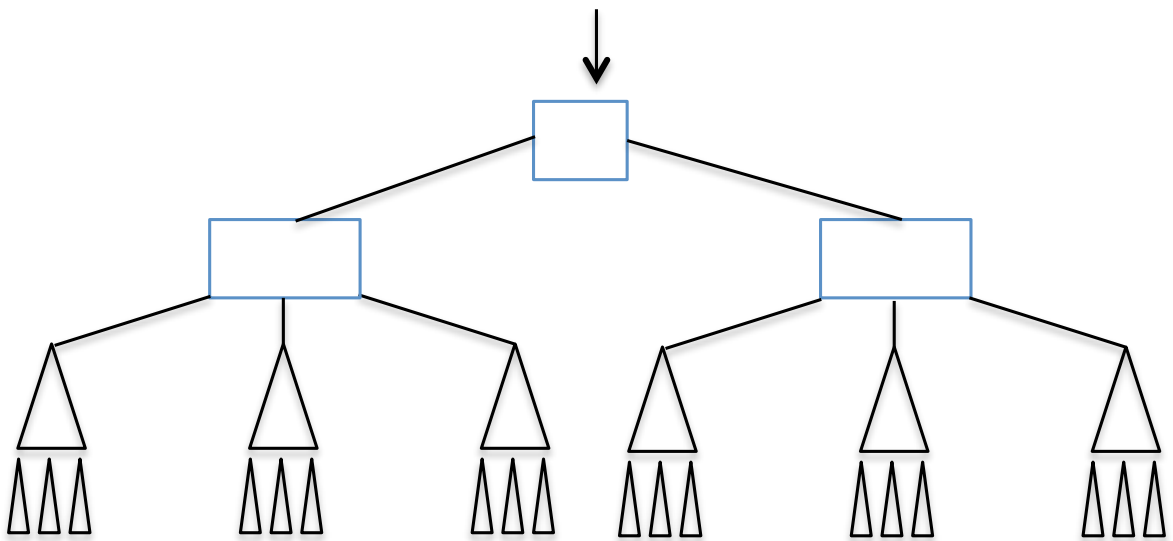


Exemple - on rajoute un niveau

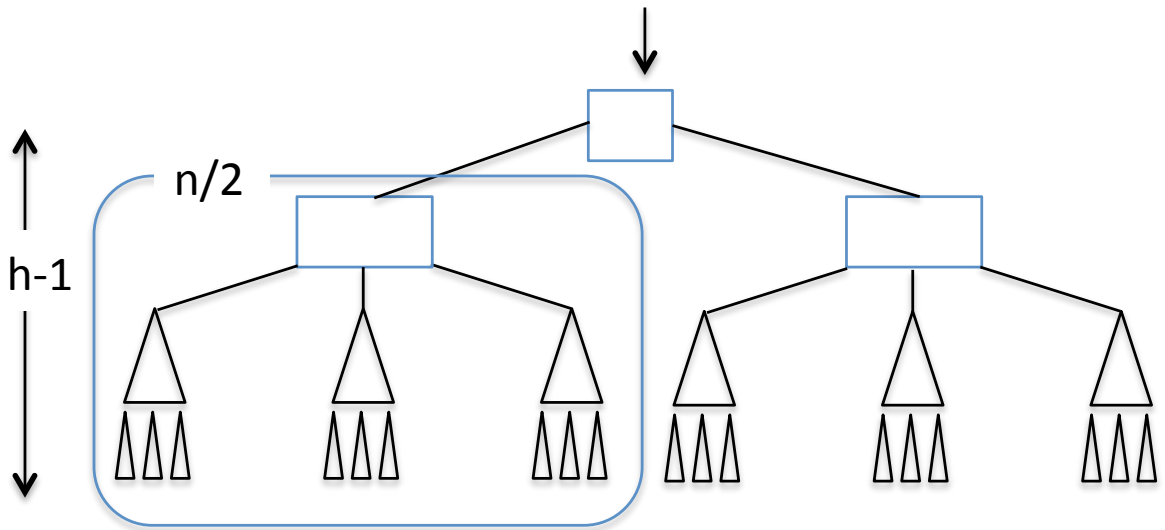


Coût

Soit n le nombre de sommets d'un Btree de degré d
h sa hauteur.
On a : $h \leq log_d \frac{n+1}{2}$



Principe : worst case



Analyse de coût

L'analyse des opérations de recherche, d'insertion et de suppression sont triviales.
Dans tous les cas, il s'agit de construire un chemin de la racine à une feuille, le coût est en $\mathcal{O}(h)$ où h est la hauteur de l'arbre.
Ainsi, la question essentielle est :

Quelle est la hauteur moyenne d'un ABR ?

ABR construits aléatoirement

Il est très difficile de connaitre des choses sur les ABR lorsqu'ils sont construits par des insertions/suppressions.
Par contre, c'est possible lorsqu'ils sont construits uniquement par des insertions.

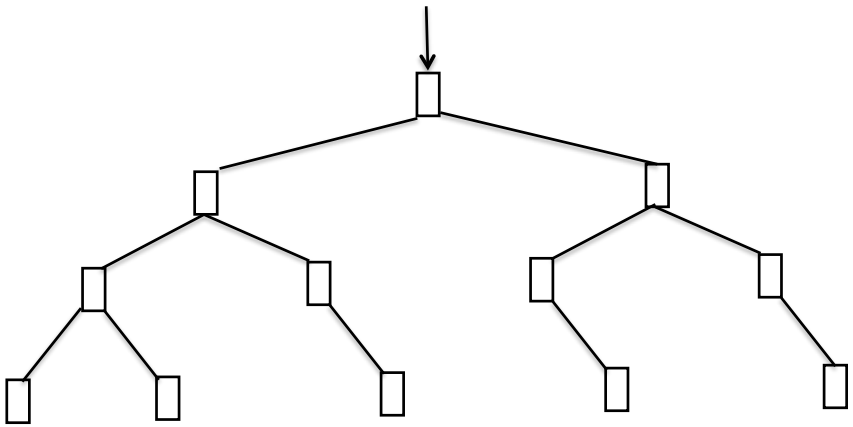
On s'intéresse aux ABR construits à partir d'un arbre vide par insertion de clés dans l'ordre aléatoire (équiprobable).

Quelques notations et rappels

Soit A un arbre binaire d'ordre n.

- La profondeur d'un noeud x est définie comme la longueur du chemin de la racine à x.
  Par convention, la profondeur de la racine est nulle.
- La profondeur (ou hauteur) de l'arbre A est le maximum des profondeurs sur ses noeuds.

Exemple (profondeur 3)



On définit un ABR construit aléatoirement par des insertions successives de n clés à partir de l'arbre vide.
On suppose que les $n!$ permutations des clés d'entrée sont équiprobables[2].

Propriété.

La hauteur d'un tel arbre est en $\mathcal{O}(logn)$.

[2] toutes les clés étant distinctes

Mais comment calculer ?

On peut résoudre ce problème de calcul de la hauteur moyenne de plusieurs façons :

- soit on détermine la hauteur d'un ABR construit aléatoirement
- soit on détermine la profondeur moyenne d'un noeud dans un ABR construit aléatoirement

Schéma de l'analyse (première méthode)

On définit la variable aléatoire X_n qui correspond à cette hauteur moyenne.
L'idée est de partir d'un choix (aléatoire) de la clé de la racine et d'insérer au fur et à mesure les clés restantes.
On note R_n le rang du choix initial parmi les n choix possibles (équiprobables).
Si $R_n = i$, l'arbre est constitué d'un sous-arbre gauche à (i-1) noeuds et d'un sous-arbre droit à (n-i) noeuds.

En pratique, on travaille plutot avec $Y_n = 2^{X_n}$. Dans ce cas, on a :

$Y_n = 2.max(Y_{i-1}, Y_{n-i})$ avec $Y_1 = 1$

Schéma de l'analyse (première méthode)

On définit la variable aléatoire X_n qui correspond à cette hauteur moyenne.
L'idée est de partir d'un choix (aléatoire) de la clé de la racine et d'insérer au fur et à mesure les clés restantes.
On note R_n le rang du choix initial parmi les n choix possibles (équiprobables).
Si $R_n = i$, l'arbre est constitué d'un sous-arbre gauche à (i-1) noeuds et d'un sous-arbre droit à (n-i) noeuds.

En pratique, on travaille plutot avec $Y_n = 2^{X_n}$. Dans ce cas, on a :

$$Y_n = 2.max(Y_{i-1}, Y_{n-i}) \text{ avec } Y_1 = 1$$

Schéma de l'analyse (première méthode)

On définit la variable aléatoire X_n qui correspond à cette hauteur moyenne.
L'idée est de partir d'un choix (aléatoire) de la clé de la racine et d'insérer au fur et à mesure les clés restantes.
On note R_n le rang du choix initial parmi les n choix possibles (équiprobables).
Si $R_n = i$, l'arbre est constitué d'un sous-arbre gauche à (i-1) noeuds et d'un sous-arbre droit à (n-i) noeuds.

En pratique, on travaille plutot avec $Y_n = 2^{X_n}$. Dans ce cas, on a :

$Y_n = 2.max(Y_{i-1}, Y_{n-i})$ avec $Y_1 = 1$

Analyse (suite)

On calcul l'espérance de cette hauteur exponentielle moyenne :

$$E(Y_n) = \frac{2}{n} \sum E(max(Y_{i-1}, Y_{n-i}))$$

$$E(Y_n) \leq \frac{2}{n} \sum (E(Y_{i-1}) + E(Y_{n-i}))$$

$$E(Y_n) \leq \frac{4}{n} \sum E(Y_i)$$

Analyse (suite)

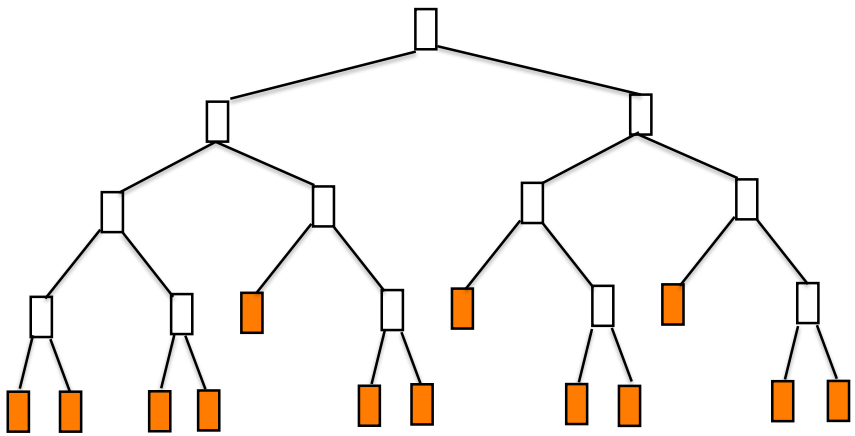
On repasse sur les X_n.

Comme $2^{E(Y_n)} \leq E(2^{X_n})$, il suffit alors de montrer que cette dernière quantité est bornée par un polynome en n.

Laisser en exercice (ou bien en consultant le livre de Cormen et al. *Introduction to Algorithms*).

Préliminaires

On définit l'*arbre complété* de A comme l'arbre dont tous les noeuds de degrés strictement inférieurs à 2 sont complétés de façon à former des sous-arbres complets.

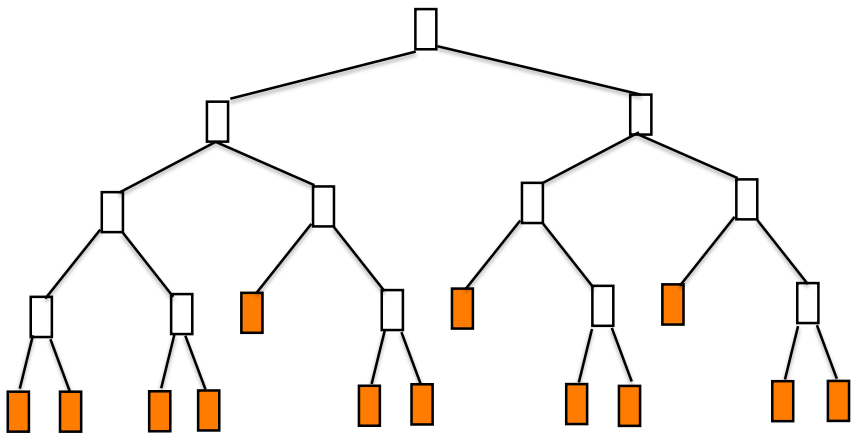


Propriété.

Le nombre de noeuds ajoutés dans l'arbre complété est $n+1$.

Préliminaires

On définit l'*arbre complété* de A comme l'arbre dont tous les noeuds de degrés strictement inférieurs à 2 sont complétés de façon à former des sous-arbres complets.



Propriété.

Le nombre de noeuds ajoutés dans l'arbre complété est $n+1$.

Propriété.

Le nombre de noeuds ajoutés dans l'arbre complété est $n+1$.

La preuve est simple par induction sur la taille de l'arbre.

Autre méthode, plus intuitive

La longueur cumulée des chemins internes de A est notée $LCI(A)$.
$LCI(A) = \sum_{x \in A} d(x, A.root)$

De même pour les noeuds externes – ceux que l'on a rajoutés pour obtenir l'arbre complété – on note $LCE(A)$ la somme des profondeurs des noeuds externes.

La profondeur moyenne de A est définie par : $\frac{LCI(A)}{n}$

Autre méthode, plus intuitive

La longueur cumulée des chemins internes de A est notée $LCI(A)$.
$LCI(A) = \sum_{x \in A} d(x, A.root)$

De même pour les noeuds externes – ceux que l'on a rajoutés pour obtenir l'arbre complété – on note $LCE(A)$ la somme des profondeurs des noeuds externes.

La profondeur moyenne de A est définie par : $\frac{LCI(A)}{n}$

Analyse

Propriétés.

$$LCI(A) = n - 1 + LCI(A.FilsG) + LCI(A.FilsD)$$
$$LCE(A) = n + 1 + LCE(A.FilsG) + LCE(A.FilsD)$$

La preuve repose sur l'argument suivant : pour chaque nœud inerne de A (sauf la racine), il faut rajouter 1 à sa profondeur dans les ss-arbres G et D pour obtenir la profondeur dans A.
On en déduit :

Propriété.

$$LCE(A) = LCI(A) + 2n$$

Cette preuve est laissée en exercice.

Calcul de la hauteur moyenne

On calcule la hauteur d'un arbre "moyen" à n noeuds insérés aléatoirement (notée C_n).
Cela correspond à la moyenne des valeurs de $LCI(A)$ sur toutes les instances.
En supposant que toutes les clés sont distinctes, il y a $n!$ permutations possibles des n clés, chacune donnant un ABR[3].

La probabilité que la première clé $x_{\sigma(1)}$ soit la $k$ième est $\frac{1}{n}$.
Dans ce cas, les sous-arbres gauches sont construits en insérant $k-1$ clés (plus petites que x_1) et les sous-arbres droits sont construits en insérant $n-k$ clés.

[3] plusieurs permutations peuvent conduire au même ABR aux symétries près...

$C_n = \frac{1}{n}\sum_{1\leq k\leq n}(C_{k-1} + C_{n-k} + (n-1))$ et $C_0 = 0$.

De même pour les noeuds externes :

$C'_n = \frac{1}{n}\sum_{1\leq k\leq n}(C'_{k-1} + C'_{n-k} + (n+1))$ and $C'_0 = 0$.

$C'_n = n+1+\frac{1}{n}\sum_{1\leq k\leq n}(C'_{k-1} + C'_{n-k})$

$C'_n = n+1+\frac{2}{n}\sum_{0\leq k\leq n-1} C'_k$

$C_n = \frac{1}{n}\sum_{1\leq k\leq n}(C_{k-1} + C_{n-k} + (n-1))$ et $C_0 = 0$.

De même pour les noeuds externes :

$C'_n = \frac{1}{n}\sum_{1\leq k\leq n}(C'_{k-1} + C'_{n-k} + (n+1))$ and $C'_0 = 0$.

$C'_n = n+1+\frac{1}{n}\sum_{1\leq k\leq n}(C'_{k-1} + C'_{n-k})$

$C'_n = n+1+\frac{2}{n}\sum_{0\leq k\leq n-1} C'_k$

Ecrivons l'expression comme:
$nC'_n = n(n+1) + 2\sum_{0 \leq k \leq n-2} C'_k + 2C'_{n-1}$
En écrivant la différence $nC'_n - (n-1)C'_{n-1}$, on déduit :
$nC'_n = 2n + (n+1)C'_{n-1}$
$\frac{C'_n}{n+1} = \frac{2}{n+1} + \frac{C'_{n-1}}{n}$
la récurrence commence en $C'_0 = 0$.

Finalement, on utilise la série harmonique $H_n = \sum_{1 \leq i \leq n} \frac{1}{i}$, on obtient :
$C'_n = 2(n+1)(H_{n+1} - 1)$.

En utilisant la propriété ($LCE(A) = LCI(A) + 2n$), on obtient :
$C_n = (n+1)(H_{n+1} - 1) - 2n$
qui est dans $\mathcal{O}(n.log(n))$ car la série harmonique est de l'ordre de $log(n)$.

De cette relation, on déduit que le cout moyen d'une recherche positive sur un arbre construit sur des insertions aléatoires est :
$\frac{C_n}{n} + 1$

C'est dans $\mathcal{O}(log(n))$.

Ce qu'il faut retenir de cette séance...

Les ABR réalisent un bon compromis pour la plupart des opérations de manipulations sur les structures de dictionnaire.

On peut améliorer un point en rajoutant une relation d'invariance supplémentaire. On verra cela la semaine prochaine avec les AVL.