IFT2015 hiver 2010 — Notes de cours

Miklós Csűrös

14 mars 2010

13 Table de symboles et arbres binaires de recherche

13.1 Table de symboles

Type abstrait «**table de symboles**» (*symbol table*) ou «dictionnaire» : ensemble d'objets avec clés. Typiquement (mais pas toujours !) les clés sont comparables (abstraction : nombres naturels).

Opérations sur un dictionnaire D :

- ⋆ search(k, D) : recherche d'un élément à clé k ← opération fondamentale — peut être fructueuse ou infructueuse.
- ⋆ insert(x, D) : insertion de l'élément x dans D

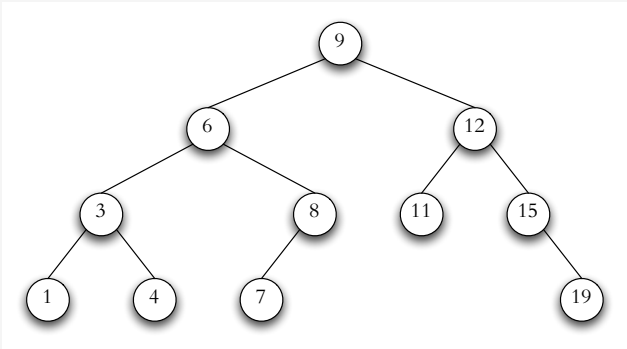
Opérations parfois supportées :

- ⋆ delete(k, D) : supprimer élément avec clé k
- ⋆ select(i, D) : sélection de l'i-ème élément (selon l'ordre des clés)

Implémentations élémentaires

- ⋆ liste chaînée ou tableau non-trié : recherche séquentielle — temps de $\Theta(n)$ au pire (même en moyenne), mais insertion/suppression en $\Theta(1)$ [si non-trié]
- ⋆ tableau trié : recherche binaire — temps de $\Theta(\log n)$ au pire, mais insertion/suppression en $\Theta(n)$ au pire cas

13.2 Arbre binaire de recherche



Dans un arbre binaire de recherche, chaque nœud a une clé.

Définition 13.1. *Un arbre binaire est un* ***arbre binaire de recherche*** *ssi les nœuds sont énumerés lors d'un parcours infixe en ordre croissant de clés.*

Accès aux nœuds

- ⋆ gauche(x) et droit(x) pour les enfants de x (null s'il n'y en a pas)
- ⋆ parent(x) pour le parent de x (null pour la racine)
- ⋆ cle(x) pour la clé de nœud x (en général, un entier dans nos discussions)

Théorème 13.1. *Soit x un nœud dans un arbre binaire de recherche. Si y est un nœud dans le sous-arbre gauche de x, alors* $\mathsf{cle}(y) \leq \mathsf{cle}(x)$. *Si y est un nœud dans le sous-arbre droit de x, alors* $\mathsf{cle}(y) \geq \mathsf{cle}(x)$.

À l'aide d'un arbre de recherche, on peut implémenter une table de symboles d'une manière très efficace.

Opérations : **recherche** d'une valeur particulière, **insertion** ou **suppression** d'une valeur, recherche de **min** ou **max**, et des autres.

Pour la discussion des arbres binaires de recherche, on va considérer les pointeurs null pour des enfants manquants comme des pointeurs vers des feuilles ou nœuds externes

Donc toutes les feuilles sont null et tous les nœuds avec une valeur cle() sont des nœuds internes.

Min et max

```
MIN(r) // nœud à clé minimale dans le sous-arbre de r
1  x ← r; y ← null
2  tandis que x ≠ null faire
3      y ← x; x ← gauche(x)
4  retourner y
```

```
MAX(r) // nœud à clé maximale dans le sous-arbre de r
1  x ← r; y ← null
2  tandis que x ≠ null faire
3      y ← x; x ← droit(x)
4  retourner y
```

Recherche. SEARCH(racine, v) retourne (a) soit un nœud dont la clé est égale à v, (b) soit null s'il n'y a pas de nœud avec clé v.

Solution récursive

```
SEARCH(x, v) // trouve clé v dans le sous-arbre de x
S1  si x = null ou v = cle(x) alors retourner x
S2  si v < cle(x)
S3  alors retourner SEARCH(gauche(x), v)
S4  sinon retourner SEARCH(droit(x), v)
```

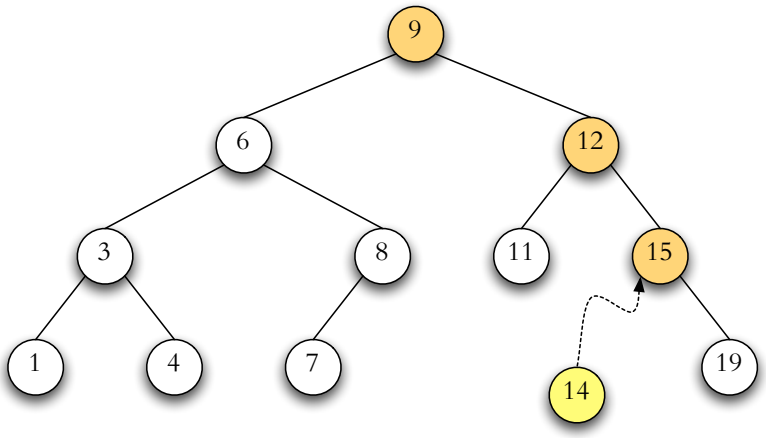
Solution itérative

```
SEARCH(x, v) // trouve clé v dans le sous-arbre de x
S1  tandis que x ≠ null et v ≠ cle(x) faire
S2      si v < cle(x)
S3      alors x ← gauche(x)
S4      sinon x ← droit(x)
S5  retourner x
```

Dans un arbre binaire de recherche de hauteur h :
MIN() prend $O(h)$
MAX() prend $O(h)$
SEARCH(racine, v) prend $O(h)$

13.3 Insertion et suppression

Insertion. On veut insérer une clé v : créer un nœud et l'ajouter à l'arbre.



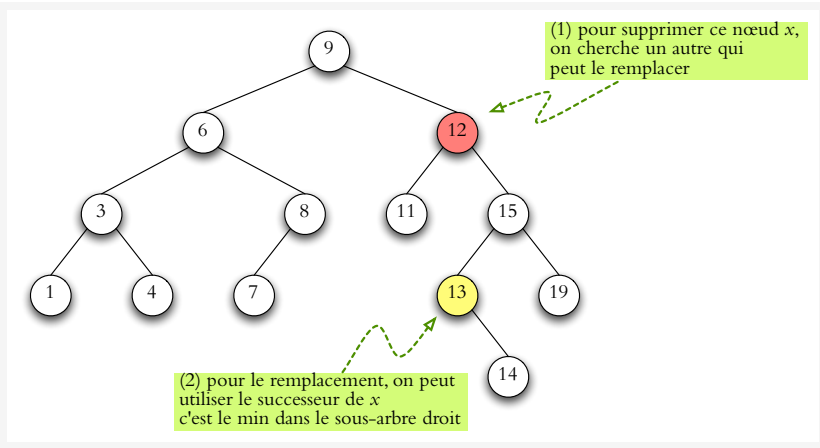
insertion de «14»

Idée : comme en SEARCH, on trouve la place pour v (enfant gauche ou droit manquant)

```
   INSERT(v) // insère la clé v dans l'arbre
I1 x ← racine
I2 si x = null alors créer nouveau nœud y avec clé v comme racine et retourner y
I3 boucler // conditions d'arrêt testées dans le corps
I4     si v = cle(x) alors retourner null // pas de valeurs dupliquées
I5     si v < cle(x)
I6     alors si gauche(x) = null
I7         alors attacher nouveau nœud y avec clé v comme enfant gauche de x et retourner y
I8         sinon x ← gauche(x)
I9     sinon si droit(x) = null
I10        alors attacher nouveau nœud y avec clé v comme enfant droit de x et retourner y
I11        sinon x ← droit(x)
```

Suppression du nœud x

1. triviale si x est n'a pas d'enfants non-null : $\mathsf{gauche}(\mathsf{parent}(x)) \leftarrow \mathsf{null}$ si x est l'enfant gauche de son parent, ou $\mathsf{droit}(\mathsf{parent}(x)) \leftarrow \mathsf{null}$ si x est l'enfant droit
2. facile si x a seulement un enfant : $\mathsf{gauche}(\mathsf{parent}(x)) \leftarrow \mathsf{droit}(x)$ si x a un enfant droit et x est l'enfant gauche de son parent (4 cas en total dépendant de la position de x et celle de son enfant)
3. un peu plus compliqué si x a deux enfants : on trouve un remplacement (successeur ou prédecesseur dans le parcours infixe)



Lemme 13.2. *Le nœud avec la clé minimale dans le sous-arbre droit de x n'a pas d'enfant gauche.*

$\Rightarrow$ il est facile d'enlever le successeur (d'un nœud à deux enfants)...

```
DELETE(z) // supprime le nœud z
D1  si gauche(z) = null ou droit(z) = null alors y ← z // cas 1. ou 2.
D2  sinon y ← MIN(droit(z)) // cas 3.
 // un des deux enfants de y est null
D3  si gauche(y) ≠ null alors x ← gauche(y) sinon x ← droit(y)
 // on enlève le nœud y de l'arbre
D4  si x ≠ null alors parent(x) ← parent(y)
D5  si parent(y) = null alors racine ← x // y était la racine
D6  sinon si y = gauche(parent(y)) // y a un parent
D7        alors gauche(parent(y)) ← x // y est enfant gauche
D8        sinon droit(parent(y)) ← x // y est enfant droit
D9  si y ≠ z alors remplacer nœud z par y dans l'arbre
```

Exercice 13.1. *Écrire l'algorithme* SUCCESSEUR(x) *qui trouve le successeur du nœud x dans le parcours infixe. Montrer que le temps de calcul de* SUCCESSEUR(x) *est $\Theta(h)$ dans le pire cas où h est la hauteur de l'arbre. Montrer que le temps de calcul est $\Theta(1)$ en moyenne (quand x est un nœud aléatoire dans l'arbre).* ***Indice*** *: considérer un parcours infixe en utilisant l'itération $x \leftarrow$* SUCCESSEUR(x).

Dans un arbre binaire de recherche de hauteur h :
INSERT(v) prend $O(h)$
suppression d'un nœud prend $O(h)$