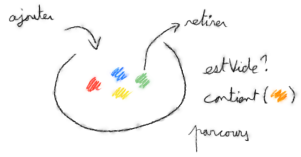
Arbres binaires de recherche et équilibrage

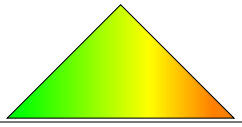
François Schwarzentruber

17 septembre 2019

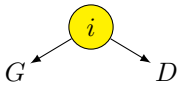
1 Arbres binaires de recherche



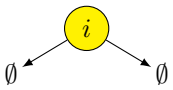
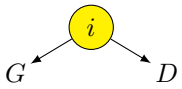
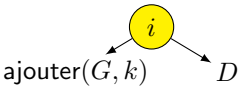
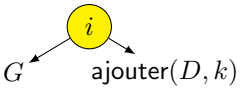
Définition. Un *arbre binaire de recherche* (ABR) est un arbre binaire où chaque nœud, est plus grand que tous les nœuds de son fils gauche, et plus petit que tous les nœuds de son fils droit.



- Entrée : un ABR A, une clef k
- Sortie : vrai ssi A contient k

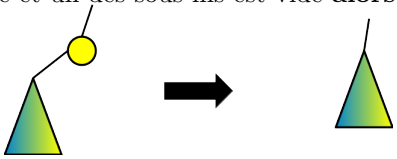
fonction contient(A, k)
  si A est l'arbre vide $\emptyset$ **alors**
    retourner faux
  sinon A est de la forme 
    si $i = k$ **alors** **retourner** vrai
    si $k < i$ **alors** **retourner** contient(G, k)
    sinon **retourner** contient(D, k)

- Entrée : un ABR A, une clef k
- Sortie : un ABR qui contient k et les éléments de A

fonction ajouter(A, k)
  si A est l'arbre vide $\emptyset$ **alors**
    retourner 
  sinon A est de la forme 
    si $i = k$ **retourner** A
    si $k < i$ **retourner** 
    sinon **retourner** 

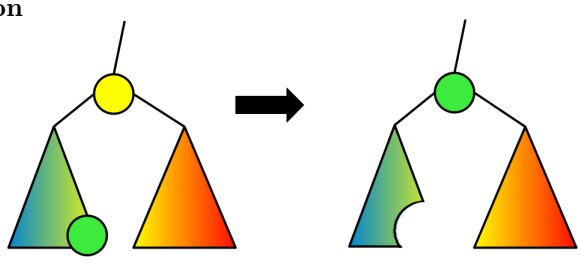
- Entrée : un ABR A, une clef k
- Sortie : un ABR qui contient les même éléments que A, sauf k

fonction supprimer(A, k) (algo informel)
  chercher k
  si k non trouvé **alors retourner** A
  si k trouvé et un des sous-fils est vide **alors**



remplacer k par l'autre sous-arbre

  sinon



supprimer le maximum dans le sous-fils gauche
le mettre à la place de k

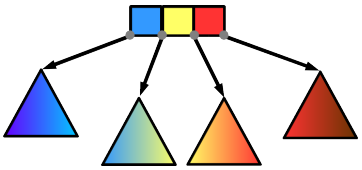
Théorème 1 *Les opérations contient, ajouter, supprimer sont en $O(\text{hauteur})$.*

La structure de données est **persistante**.

2 Équilibrage : B-arbres

Définition. Un *B-arbre* est un arbre qui vérifie, en plus des conditions d'ABR, les conditions suivantes :

1. Tout nœud a au plus $2B-1$ clefs ;
2. Tout nœud non racine a au moins $B-1$ clefs ;
3. Si l'arbre est non vide, la racine est non vide ;
4. Un nœud qui possède k fils contient $k-1$ clefs ;
5. Les feuilles sont à la même profondeur h.



Un nœud est dit *complet* lorsqu'il a exactement $2B-1$ clefs.

Théorème 2 *Un B-arbre de hauteur h, non vide, qui contient n clefs est tel que*

$$h \leq \log_B \left(\frac{n+1}{2} \right).$$

Démonstration. Au niveau 0, il y a 1 noeud, qui contient au moins une clef car l'arbre est non vide. Chaque nœud (sauf la racine), contient au $(B-1)$ clefs. Ainsi, si la racine a des fils, elle a deux fils au moins. Chaque nœud non racine, qui n'est pas une feuille a au moins B fils. On obtient donc la table suivante :

au niveau ...	on a au moins ... nœuds	et au moins ... clefs
0	1	1
1	2	$2(B-1)$
2	$2B$	$2B(B-1)$
3	$2B^2$	$2B^2(B-1)$
$\vdots$	$\vdots$	$\vdots$
h	$2B^{h-1}$	$2B^{h-1}(B-1)$

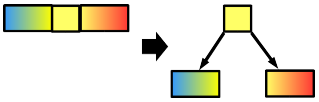
D'où

$$n \geq 1 + 2(B-1)\sum_{i=0}^{h-1} B^i = 1 + 2(B-1)\frac{B^h-1}{B-1} = 1 + 2(B^h-1) = 2B^h - 1.$$

On prouve le théorème en passant au log. ■

- Entrée : un B-arbre A, une clef k
- Sortie : un B-arbre contenant k en plus

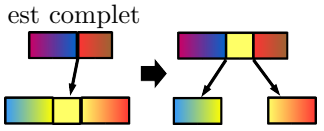
fonction insererDepuisRacine(A, k)
  si A complet **alors**
    éclater la racine A



  insererDepuisNoeudIncomplet(A, k)

- Entrée : un nœud A non complet, une clef k
- Sortie : tout l'arbre est un B-arbre contenant k en plus

fonction insererDepuisNoeudIncomplet(A, k)
  si A est une feuille **alors** on insère la clef k dans A
  sinon
    S := sous-arbre de A correspondant à k
    si S est complet

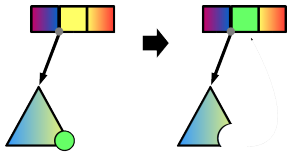


éclater S
remonter l'élément central de S dans A

    S := le sous-arbre de A correspondant à k
    insererDepuisNoeudIncomplet(S, k)

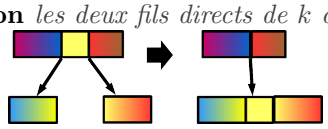
- Entrée : un nœud A qui est soit la racine ou qui contient $\geq B$ clefs, une clef k
- Sortie : tout l'arbre est un B-arbre mais k n'y est plus

fonction supprimer(A, k)
  si le nœud A contient la clef k **alors**
    si A est une feuille **alors** supprimer k de A
    sinon si un des fils directs de k contient plus de B clefs



$S :=$ un de ces fils
supprimer le min/max dans S
remplacer k par ce min/max

    sinon *les deux fils directs de k ont $B-1$ clefs*

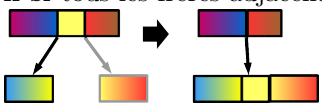


enlever k de A
créer le fils avec les clefs des deux fils directs et k que l'on connecte à A
si A est devenu vide **alors** effacer A
supprimer(ce fils, k)

  sinon *//A ne contient pas la clef k*
    $S :=$ le sous-fils de A correspondant à k
    si S a $B-1$ clefs
      si un des frères adjacents de S a plus de B clefs

remonter un des éléments de ce frère dans A
descendre une clef de A dans S

      sinon si tous les frères adjacents de S ont $B-1$ clefs

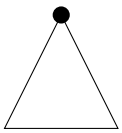


$S :=$ fusion(S, la valeur de A entre S et ce frère, ce frère)
      si A est devenu vide **alors** effacer A
    supprimer(S, k)

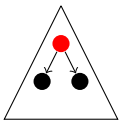
3 Arbres rouge-noir

Définition 3 *Un arbre rouge-noir est un ABR où les sommets sont peints en rouge ou noir, et tel que :*

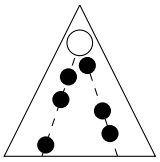
1. La racine est noire ;



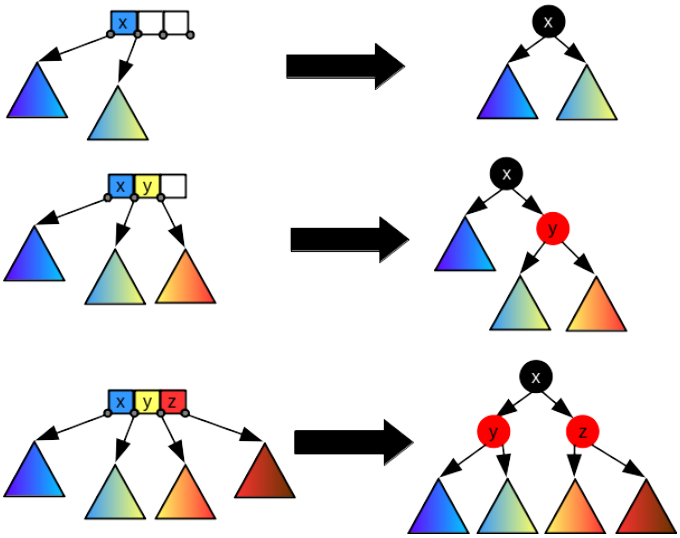
2. Les fils d'un nœud rouge sont noirs ;



3. Toutes les branches partant d'un certain nœud vers un arbre vide $\emptyset$ possèdent le même nombre de nœuds noirs.



Un arbre rouge-noir représente un B-arbre avec $B = 2$; les opérations sur les arbres rouge-noir simulent les opérations sur les B-arbres.



4 Discussion

- requête du type “donne moi les éléments ≤ 10”
- garanties des complexités en $O(\log n)$
- structure de données persistante
- prend moins de mémoire qu’une table de hachage si peu d’éléments
- beaucoup d’accès E/S, en pratique plus lent qu’une table de hachage (si elle est suffisamment grande)
- il faut implémenter $\leq$

5 Notes bibliographiques

Plusieurs chercheurs ont inventés les arbres binaires de recherche, voir une version de l’histoire dans [Knu73]. L’idée d’équilibrage d’ABR vient des travaux de Georgii Adelson-Velsky et Evguenii Landis [AVL], (arbres AVL). Bayer and McCreight ont inventé les B-arbres [BM70]. Bayer invente les arbres rouge-noir [Bay72], bien que la convention rouge rouge est proposé par Guibas and Sedgewick [GS78]. Les arbres $B+$ généralisent les B-arbres. Un travail de 2018 à l’IRISA [SAA+18] permettent l’accès rapide à une base de données chiffrée en s’inspirant des arbres $B+$.

References

[AVL] Adelson-Velsky and EM Landis. An algorithm for the organization of information.

[Bay72] Rudolf Bayer. Symmetric binary b-trees: Data structure and maintenance algorithms. *Acta Informatica*, 1:290–306, 1972.

[BM70] Rudolf Bayer and Edward M. McCreight. Organization and maintenance of large ordered indexes. In *Record of the 1970 ACM SIGFIDET Workshop on Data Description and Access, November 15-16, 1970, Rice University, Houston, Texas, USA (Second Edition with an Appendix)*, pages 107–141. ACM, 1970.

[GS78] Leonidas J. Guibas and Robert Sedgewick. A dichromatic framework for balanced trees. In *19th Annual Symposium on Foundations of Computer Science, Ann Arbor, Michigan, USA, 16-18 October 1978*, pages 8–21. IEEE Computer Society, 1978.

[Knu73] Donald E. Knuth. *The Art of Computer Programming, Volume III: Sorting and Searching*. Addison-Wesley, 1973.

[SAA+18] Cetin Sahin, Tristan Allard, Reza Akbarinia, Amr El Abbadi, and Esther Pacitti. A differentially private index for range query processing in clouds. In *34th IEEE International Conference on Data Engineering, ICDE 2018, Paris, France, April 16-19, 2018*, pages 857–868. IEEE Computer Society, 2018.