Chapitre 4

Recherche monodimensionnelle

4.1 Dictionnaires

Un dictionnaire est une structure de données permettant de rechercher, insérer ou supprimer des données. Chaque donnée est supposée posséder une clé qui l'identifie. Cette clé doit appartenir à un univers totalement ordonné (typiquement des entiers). Pour ranger les données on se sert de leur clé. Par la suite on ne s'intéresse qu'aux clés, les "véritables" données pouvant être obtenues à partir des clés à l'aide d'un pointeur par exemple. D'autres opérations telles que la recherche de la clé minimale ou maximale, de la clé suivant ou précédant une clé donnée sont possibles. La fusion et la scission de dictionnaires sont également des opérations courantes.

Classiquement, mais pas uniquement (voir plus bas) les dictionnaires sont représentés à l'aide d'arbres. Ces arbres peuvent être binaires (comme pour les arbres *AVL* ou *bicolores)* ou non (*arbres a-b*).

Références :

http ://www.cs.sunysb.edu/∼algorith/

http ://www.nist.gov/dads/

Introduction to Algorithms. Cormen, Rivest, Leiserson and Stein, M.I.T. PRESS, second edition 2001. Version française chez Dunod, 1994.

STL (http ://www.sgi.com/tech/stl/)

LEDA (http ://www.mpi-sb.mpg.de/LEDA/MANUAL/MANUAL.html)

4.1.1 Arbres binaires de recherche

Définition 4.1 *Un* arbre binaire de recherche *est un arbre binaire dont les noeuds possèdent des clés rangées dans l'ordre infixe, i.e. un parcours dans l'ordre infixe de l'arbre (sag(b).racine(b).sad(b)) visite les clés dans l'ordre croissant.*

Propriété : L'ensemble des arbres binaires de recherche est stable par les opérations de

rotation.

Note : parfois on ne range les clés qu'aux noeuds externes et un noeud interne contient une valeur intermédiaire entre celles de son sous-arbre gauche et de son sous-arbre droit.

4.2 Structures randomisées

Les structures classiques telles que les arbres AVL, bicolores ou a-b sont des structures déterministes. Les performances des opérations de dictionnaires ne dépendent ni des données ni de l'ordre de construction de ces structures. Les algorithmes de construction et de modification sont cependant relativement compliqués. Les structures randomisées telles que les *skip lists* ou les *treaps* utilisent des générateurs aléatoires dans leur construction et permettent d'obtenir des performances *en moyenne* équivalentes à celles des structures déterministes. Les constructions sont généralement plus simples.

4.2.1 Skip list

Une structure de *skip list* sur un ensemble M de n clés est un dictionnaire sur M construit de la manière suivante :

On tire *au hasard et de manière indépendante* chaque clé de M avec une probabilité $1/2$. On obtient ainsi un sous-ensemble M_2 de M avec lequel on recommence la procédure de tirage. On continue ainsi jusqu'à obtenir l'ensemble vide. On obtient finalement une *gradation* de M, c'est à dire une suite décroissante de sous-ensembles de M :

$$M = M_1 \supseteq M_2 \supseteq \ldots \supseteq M_r \supset M_{r+1} = \emptyset$$

Une skip list sur une telle gradation s'obtient à partir des r listes triées de chaque M_i, appelé niveau, augmenté pour chaque clé de la liste M_i d'un pointeur vers l'occurrence de cette clé dans la liste M_{i-1} et d'un pointeur vers son successeur dans la liste M_i. On ajoute également un élément fictif minimal dans chaque liste que l'on relie entre-eux. On s'intéresse à la taille de cette structure et à la complexité des opérations de recherche, insertion, suppression en fonction du nombre n de clés. Puisque ces grandeurs dépendent de tirages aléatoires, on s'intéresse à leur valeur moyenne où l'on considère tous les tirages de clés indépendants. De plus, pour caractériser le fait que les grandeurs s'écartent très peu des valeurs moyennes (c.a.d. que la distribution des valeurs est bien localisée autour de la moyenne) on introduit la notation suivante :

Définition 4.2 *Soient $f(n)$ et $g(n)$ des variables aléatoires dépendant d'un paramètre n. On écrit $f = \tilde{O}(g)$ si $P(f(n) > cg(n)) < 1/p(n,c)$ où $p(n,c)$ est un polynôme en n dont le degré tend vers l'infini avec c.*

Intuitivement, un $\tilde{O}(g)$ est un $O(g)$ avec très forte probabilité.

Soit h_i la variable aléatoire qui donne le nombre de niveaux auxquels appartient la clé i. h_i suit une loi géométrique de paramètre $1/2$, d'où $P(h_i = k) = 1/2^k$ et $E(h_i) = 2$.

On pose $h = \max_{1 \leq i \leq n} h_i$ la hauteur d'une skip list.

Lemme 4.3 *$E(h) = O(\log n)$ et $h = \tilde{O}(\log n)$.*

Preuve :

$$P(h > k) = P((h_1 > k) \vee (h_2 > k) \vee \ldots \vee (h_n > k)) \leq \sum_{i=1}^{n} P(h_i > k) = n/2^k.$$

On en déduit pour tout $c > 0$ que $P(h > c \log n) \leq 1/n^{c-1}$, d'où $h = \tilde{O}(\log n)$. On a de plus (cf. lemme 1.9)

$$E(h) = \sum_{k=0}^{c \log n - 1} P(h > k) + \sum_{k=c \log n}^{\infty} P(h > k) \leq c \log n + n \sum_{k=c \log n}^{\infty} \frac{1}{2^k} = c \log n + \frac{2}{n^{c-1}}.$$

□

On note $t = \sum_{1 \leq i \leq n} h_i$ la taille d'une skip list.

Lemme 4.4 *La taille t d'une skip list sur n clés vérifie $E(t) = O(n)$ et $t = \tilde{O}(n)$.*

Preuve : Par linéarité de l'espérance, on a $E(t) = \sum_i E(h_i) = 2n$. En mettant bout a bout les tirages pour chaque clé, t s'interprète comme le nombre de tirages nécessaires pour obtenir n échecs. Dit autrement t suit une loi binomiale négative de paramètres n et $1/2$. Le lemme 1.28 utilisant la technique de majoration de Chernoff permet de conclure. □

Remarque : La construction d'une skip list sur un ensemble de n clés peut s'obtenir après tri de ses clés en temps proportionnel à sa taille, i.e. en temps $O(n \log n + t) = \tilde{O}(n \log n)$.

Soit K une clé, fixée une fois pour toute, à rechercher dans une skip list. On note K_i la plus grande clé du niveau i (i.e. de M_i) inférieure ou égale à K. On note également X_i le nombre de clés comprises, au sens large, entre K_{i+1} et K_i dans M_i. Lorsque M_i est vide, on pose $X_i = 0$ et $K_i =$ l'élément fictif minimal (voir plus haut). Pour rechercher K dans la skip list on commence par parcourir les clés du plus haut niveau M_r dans l'ordre croissant jusqu'à atteindre K_r. À l'aide du pointeur de K_r vers le niveau $r-1$ on descend sur sa copie dans M_{r-1} puis on parcourt M_{r-1} dans l'ordre croissant des clés jusqu'à atteindre K_{r-1}. La procédure se poursuit récursivement jusqu'au premier niveau. La recherche est fructueuse si et seulement si $K_1 = K$.

Lemme 4.5 *Le temps de recherche d'une clé fixée est un $\tilde{O}(\log n)$ et le temps moyen est un $O(\log n)$.*

Preuve : Le temps de recherche est clairement proportionnel à la longueur ℓ du "chemin" de recherche, i.e. à $\ell = \sum_{i \geq 1} X_i$ (cette longueur inclut les marches horizontales et verticales).

Notons que pour M_i fixé, M_{i+1} est obtenu en tirant aléatoirement et indépendamment chaque clé de M_i avec une probabilité $1/2$. Par conséquent, la variable aléatoire $X_i|M_i$ suit une loi géométrique de paramètre $1/2$ (plus précisément $X_i|M_i$ est majorée par une variable suivant une telle loi). On en déduit $E(X_i|M_i) \leq 2$, d'où inconditionnellement $E(X_i) \leq 2$.

On note Y_i la variable aléatoire valant 0 si M_i est vide et 1 sinon, de sorte que $X_i \leq nY_i$. On a $P(Y_i = 1) = P(h \geq i) \leq n/2^{i-1}$, soit $E(Y_i) \leq n/2^{i-1}$. On obtient

$$E(\sum_{i\geq 1} X_i) = \sum_{i\geq 1} E(X_i) \leq \sum_{i=1}^{c\log n} E(X_i) + \sum_{i>c\log n} E(nY_i) \leq 2c\log n + 2/n^{c-2}.$$

Et on conclut $E(\ell) = O(\log n)$.

Montrons que $\ell = \tilde{O}(\log n)$. Pour cela on 'coupe' ℓ en deux en écrivant $\ell = \ell_{\leq} + \ell_{>}$ avec $\ell_{\leq} = \sum_{i=1}^{c\log n} E(X_i)$ et $\ell_{>} = \sum_{i>c\log n} E(X_i)$. Alors $\ell_{\leq}$ est une somme de $c\log n$ variables aléatoires majorées par des variables de loi géométrique de paramètre $1/2$ et est donc majorée par une variable aléatoire suivant une loi binomiale négative de paramètres $c\log n$ et $1/2$. On en déduit par la technique de Chernoff (lemme 1.28) que

$$P(\ell_{\leq} > c(2+d)\log n) \leq \exp(-dc\log n/4) = O(\frac{1}{n^{dc/4}})$$

dès que $d \geq 3$. D'où $\ell_{\leq} = \tilde{O}(\log n)$.

Par ailleurs $\ell_{>}$ est trivialement majorée par la hauteur h de la skip list additionnée au nombre d'éléments restant au niveau $c\log n + 1$. Soit Z_i la fonction indicatrice de la présence de la i-ème clé au niveau $c\log n + 1$, de sorte que $\ell_{>} \leq h + \sum_{i=1}^{n} Z_i$. Les Z_i sont indépendantes et suivent une loi de Bernoulli de paramètre $1/2^{c\log n} = 1/n^c$. Donc $\sum_{i=1}^{n} Z_i$ suit une loi binomiale de paramètre n et $1/n^c$. Par l'inégalité de Markov on a

$$P(\sum_{i=1}^{n} Z_i > c) \leq E(\sum_{i=1}^{n} Z_i)/c \leq \frac{1}{cn^{c-1}}.$$

D'où $\sum_{i=1}^{n} Z_i = \tilde{O}(1)$. D'après le lemme 4.3, $h = \tilde{O}(\log n)$ et on conclut finalement que $\ell_{>} = \tilde{O}(\log n)$ puis que $\ell = \tilde{O}(\log n)$. □

On a en fait un résultat plus fort ne dépendant pas de la clé particulière à chercher :

Lemme 4.6 *Le temps maximal de recherche d'une clé quelconque dans une skip list est un $\tilde{O}(\log n)$.*

Preuve :

$$P(\max_K \ell(K) > c\log n) \leq \sum_K P(\ell(K) > c\log n) \leq \frac{n}{p(n,c)} = \frac{1}{q(n,c)}.$$

□

Lemme 4.7 *Le temps maximal de suppression d'une clé quelconque dans une skip list est un* $\tilde{O}(\log n)$.

Pour supprimer une clé on effectue sa recherche en la supprimant des niveaux M_i où elle apparaît. Le temps de suppression est donc de l'ordre du temps de recherche. Si on dispose d'un pointeur sur l'élément à supprimer alors la suppression est un $\tilde{O}(1)$ car $h_K = \tilde{O}(1)$.

Lemme 4.8 *Le temps maximal d'insertion d'une clé dans une skip list est un* $\tilde{O}(\log n)$.

Pour insérer un clé on effectue des tirages jusqu'à obtenir un échec. Si k est le nombre de tirages effectués on insère la clé dans les k premiers niveaux en même temps que l'on effectue une recherche dans la skip list.

Référence :

- Skip Lists : A probabilistic Alternative to Balanced Trees. W. Pugh. Communication of the ACM, 33(6), june 1990.

4.2.2 Arbres binaires de recherche aléatoires

Étant donnée une permutation σ de $[1, n]$ on construit un arbre binaire de recherche en introduisant $\sigma(1)$ puis $\sigma(2)$, ..., puis $\sigma(n)$ dans un arbre initialement vide. Les $n!$ permutations de $[1, n]$ permettent d'obtenir toutes les formes possibles d'arbres binaires de recherche sur n éléments. En considérant une loi de probabilité uniforme sur les $n!$ permutations on en déduit une loi de probabilité sur les arbres binaires de recherche.

Proposition 4.9 *La hauteur d'un arbre binaire de recherche aléatoire est un* $\tilde{O}(\log n)$ *et sa profondeur moyenne est un* $O(\log n)$.

Notons que d'après le théorème 1.31, la hauteur moyenne d'un arbre de recherche aléatoire est un $O(\log n)$, ce qui implique évidement que la profondeur moyenne des clés est du même ordre. On donne ici une preuve de ce dernier résultat, moins fort mais plus simple à montrer.

Preuve : On note X_j^i la variable aléatoire indiquant si j est un ancêtre de i dans l'arbre binaire de recherche associé à une permutation aléatoire σ. On note τ la permutation (aléatoire) inverse de σ. $X_j^i = 1$ si et seulement si $\tau(j) = \min_{k\in[i,j]} \tau(k)$, c.a.d. si et seulement si j est inséré avant i et aucun élément inséré avant j ne sépare i et j. Clairement $P(X_j^i = 1 \mid \tau([i, j])) = \frac{1}{|i-j|+1}$, d'où $E(X_j^i) = \frac{1}{|i-j|+1}$.

Soit $h_i = \sum_{j\neq i} X_j^i$ la hauteur de i dans l'arbre binaire de recherche. Par linéarité de l'espérance $E(h_i)$ vaut $\sum_{j=1}^{n} \frac{1}{|i-j|+1} = H_i + H_{n+1-i} - 1 = O(\log n)$.

Par ailleurs, pour i fixé et $i < j$ le lemme 4.10 ci-dessous indique que les variables X_j^i sont mutuellement indépendantes. Par la technique de Chernoff on peut donc écrire pour tout t et tout λ positif

$$P(\sum_{j>i} X_j^i > t) \leq \frac{\prod_{j>i} E(\exp(\lambda X_j^i))}{\exp(\lambda t)}.$$

Or

$$E(\exp(\lambda X_j^i)) = \frac{\exp(\lambda)}{j-i+1} + 1 - \frac{1}{j-i+1} = 1 + \frac{\exp(\lambda)-1}{j-i+1} \leq \exp(\frac{\exp(\lambda)-1}{j-i+1})$$

d'où

$$P(\sum_{j>i} X_j^i > t) \leq \exp((\exp(\lambda)-1)H_n - \lambda t) \leq \exp((\exp(\lambda)-1)(\ln n + 1) - \lambda t).$$

En choisissant $t = c \ln n$ et $\lambda = \ln c$ on trouve

$$P(\sum_{j>i} X_j^i > c \ln n) \leq \frac{\exp(c-1)}{n^{c(\ln c - 1)+1}} = \frac{1}{p(n,c)}.$$

En opérant de même avec la somme $\sum_{j<i} X_j^i$ on en déduit que $h_i = \tilde{O}(\log n)$. Comme pour l'analyse de la hauteur d'une skip list on conclut finalement que la hauteur $h = \max_i h_i$ d'un arbre binaire de recherche aléatoire est un $\tilde{O}(\log n)$ et que son espérance est un $O(\log n)$. □

Dit autrement un arbre binaire de recherche aléatoire est presque toujours bien équilibré.

Lemme 4.10 *Avec les notations de la preuve ci-dessus, soient* $1 \leq i < j_1 < j_2 < \ldots < j_k \leq n$. *Alors les variables* $X^i_{j_1}, X^i_{j_2} \ldots, X^i_{j_k}$ *sont mutuellement indépendantes. On a un résultat analogue pour* $1 \leq j_1 < j_2 < \ldots < j_k < i \leq n$.

Preuve : On raisonne par récurrence sur k. Soient $\epsilon_r \in \{0,1\}$, $r \in [1,k]$. On pose $A = \{X^i_{j_1} = \epsilon_1 \wedge \ldots \wedge X^i_{j_{k-1}} = \epsilon_{k-1}\}$. Par hypothèse de récurrence à l'ordre $k-1$, les variables $X^i_{j_1}, X^i_{j_2} \ldots, X^i_{j_{k-1}}$ sont indépendantes. On remarque en particulier que $P(A) = \prod_{r=1}^{k-1} P(X^i_{j_r} = \epsilon_r)$ ne dépend que des grandeurs $|j_r - i|$. On considère une partie $I \subset [1,n]$ à $j_k - i + 1$ éléments et le sous-ensemble de permutations de $[1,n]$:

$$B_I = \{\tau : \tau([i, j_k]) = I \wedge \tau(j_k) = \min I\}$$

On note que les B_I sont disjoints et que leur union, $\cup_I B_I$, est l'événement $\{X^i_{j_k} = 1\}$. J'affirme que $P(A|B_I) = P(A)$. Pour le voir on considère l'injection croissante $\iota : [1, j_k - i] \to [1,n]$ telle que $\mathrm{Im}\iota = I$ et la surjection ϕ de B_I dans les permutations de $[1, j_k - i]$:

$$\begin{array}{ccc} B_I & \stackrel{\phi}{\to} & \mathcal{S}_{j_k - i} \\ \tau & \mapsto & \tau' : \ell \mapsto \iota^{-1}(\tau(\ell + i - 1)) \end{array}$$

Puisque l'appartenance de τ à A ne dépend que des ordres relatifs de $\tau(i), \tau(i+1), \ldots, \tau(j_{k-1})$, on a

$$\tau \in A \Leftrightarrow \phi(\tau) \in A' = \{X^i_{j_1 - i + 1} = \epsilon_1, X^i_{j_2 - i + 1} = \epsilon_2, \ldots, X^i_{j_{k-1} - i + 1} = \epsilon_{k-1}\}$$

De plus, le nombre $|\phi^{-1}(\tau')|$ ne dépend pas de $\tau' \in \mathcal{S}_{j_k - i}$. On en déduit, avec la remarque ci-dessus, $P(A|B_I) = P(A') = P(A)$, et par suite $P(A|X^i_{j_k} = 1) = P(A)$ (cf. exercice 1.20). En reportant cette égalité dans la suivante

$$P(A) = P(A|X^i_{j_k} = 1)P(X^i_{j_k} = 1) + P(A|X^i_{j_k} = 0)P(X^i_{j_k} = 0)$$

on obtient également $P(A|X^i_{j_k} = 0) = P(A)$. Finalement on en déduit,

$$\forall \epsilon_k \in \{0,1\} : P(A \wedge X^i_{j_k} = \epsilon_k) = P(A)P(X^i_{j_k} = \epsilon_k)$$

soit encore

$$P(X^i_{j_1} = \epsilon_1 \wedge \ldots \wedge X^i_{j_k} = \epsilon_k) = \prod_{r=1}^{k} P(X^i_{j_r} = \epsilon_r)$$

□

4.2.3 Tree + Heap = Treap

Introduits sous le nom d'arbres cartésiens par J. Vuillemin, les treaps sont des arbres binaires de recherches dont les noeuds possèdent une clé et une priorité comprise entre 0 et 1. Un treap est un arbre binaire de recherche pour les clés - les noeuds dans le sous arbre gauche (droit) d'un noeud ont une clé inférieure (supérieure) à la clé de ce noeud - et un tas pour les priorités (le parent d'un noeud a une priorité plus grande que celle de ce noeud). Un treap est donc le résultat d'un arbre binaire de recherche construit en introduisant un à un les éléments dans l'ordre de leur priorité. En choisissant les priorités de manière aléatoire on peut simuler un arbre binaire de recherche aléatoire. Ceci assure d'après le lemme 4.9 que la recherche a un coût en $\tilde{O}(\log n)$. Il est possible de maintenir dynamiquement un treap, i.e. d'insérer ou de supprimer des éléments en conservant la propriété d'arbre aléatoire. Pour cela, on utilise le fait que l'opération de rotation sur un arbre binaire (cf. section 1.4) préserve l'ordre infixe, intervertit la parenté pour deux noeuds, et préserve la propriété de tas pour le reste. Ceci permet, par rotations successives de faire remonter ou descendre un noeud dans un treap pour l'ôter ou le positionner à sa bonne place, c'est-à dire comme s'il avait été introduit 'à l'instant' donné par sa priorité.

Références :

- Randomized search trees. C. Aragon and R. Seidel. Algorithmica 16. pp 464-497. 1996.

- Randomized Binary Search Trees. C. Martínez and S. Roura. Journal of the ACM. 45. pp 288-323. 1998.