Algorithmique et Programmation

TD n° 4 : Hachage et tas

École normale supérieure – Département d'informatique
algoL3@di.ens.fr

2014-2015

Exercice 1. Hachage avec chaînage

Soient $\mathcal{U}$ un univers, $S \subset \mathcal{U}$ et m un entier.

1. Considérons une table de hachage avec chaînage pour l'ensemble S de cardinal $\#S = n$ construite avec une fonction de hachage h tirée uniformément aléatoirement parmi toutes les fonctions de $\mathcal{U} \longrightarrow \{0, 1, \ldots, m-1\}$.
   Montrer que, pour $m = n$, la longueur de la liste chaînée la plus longue dans la table de hachage est de l'ordre de $O(\log n / \log \log n)$ avec probabilité au moins $1 - n^{-1}$ (sur le choix de la fonction de hachage).
2. Supposons désormais que nous utilisons deux fonctions de hachage h_1 et h_2 tirées uniformément aléatoirement et indépendemment parmi toutes les fonctions de $\mathcal{U} \longrightarrow \{0, 1, \ldots, m-1\}$ et qu'un élément x est inséré dans la table de hachage à la position $h_1(x)$ ou $h_2(x)$ qui contient le moins d'éléments déjà hachés. Donner un argument heuristique pour affirmer que la longueur de la liste chaînée la plus longue est de l'ordre de $O(\log \log n)$ avec une bonne probabilité.

Exercice 2. Fonctions de hachage coucou

Soient $\mathcal{U}$ un univers, $S \subset \mathcal{U}$ et m un entier. Nous utilisons deux fonctions de hachage h_1 et h_2 tirées uniformément aléatoirement et indépendemment parmi toutes les fonctions de $\mathcal{U} \longrightarrow \{0, 1, \ldots, m-1\}$. Pour insérer un élément x dans la table de hachage, nous calculons $h_1(x)$ et $h_2(x)$ et si l'une des deux positions est vide, nous y plaçons x. Dans le cas contraire, nous enlevons l'élement y présent dans la case $h_1(x)$ et nous le remplaçons par x puis nous plaçons y dans son autre position possible. Si cette position est occupée par un élément z, nous déplaçons z dans sa position alternative et ainsi de suite. Si le processus échoue (*i.e.* si nous déplaçons deux fois un même élément), nous essayons de placer x à la position $h_2(x)$ de la même façon. Si ce processus échoue également, alors la table de hachage est totalement reconstruite. Nous supposons que $n = \#S \leq m/4$.

1. Donner la complexité dans le pire cas des opérations de suppression et de recherche.
2. Considérons le graphe (dit *graphe coucou*) dont les sommets forment l'ensemble $V = \{0, \ldots, m-1\}$ et les arêtes sont les paires $\{h_1(x), h_2(x)\}$ pour $x \in S$. Montrer que la table de hachage est reconstruite si et seulement si dans ce graphe il existe un ensemble de k sommets tel que le graphe induit a $k+1$ arêtes.
3. Montrer que pour deux sommets i et j du graphe coucou et pour tout entier $\ell \geq 1$, la probabilité qu'il existe un plus court chemin de longueur ℓ de i à j est inférieure à $2^{-\ell}/m$.
4. Montrer que la probabilité de l'événement de la question 2 est majorée par $O(1/n)$.
5. En déduire que le coût amorti de l'opération d'insertion est de l'ordre de $O(1)$.

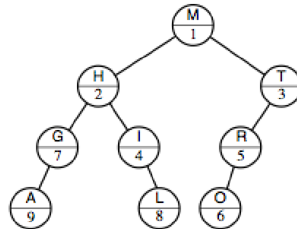
Exercice 3. Hachage parfait

Soient $\mathcal{U}$ un univers, $S \subset \mathcal{U}$ et m un entier. Une fonction de hachage parfaite h pour $(\mathcal{U}, S, m)$ et une fonction $h : \mathcal{U} \longrightarrow \{0, 1, \ldots, m-1\}$ qui est injective sur S.

1. Montrer que si une famille de fonctions de hachage $\mathcal{H} = \{h_k : \mathcal{U} \longrightarrow \{0, 1, \ldots, m-1\}\}_{k \in \mathcal{K}}$ est universelle, et si $S \subset \mathcal{U}$ vérifie $\#S = n$ avec $n^2 \leq m$, alors au moins la moitié des fonctions de $\mathcal{H}$ sont des fonctions de hachage parfaite pour $(\mathcal{U}, S, m)$.
2. En déduire un moyen d'implanter une table de hachage où les opérations d'insertion, de recherche et de suppression demandent un temps $O(1)$ et qui nécessite un espace de taille $O(n^2)$.
3. Considérons une table de hachage à deux niveaux où le premier niveau utilise une fonction de hachage $h : \mathcal{U} \longrightarrow \{0, 1, \ldots, m-1\}$ avec $m = O(n)$ et où pour chaque case $i \in \{0, 1, \ldots, m-1\}$ une deuxième fonction de hachage $h_i : \mathcal{U} \longrightarrow \{0, 1, \ldots, m_i\}$ est utilisée avec $m_i = O(n_i^2)$ où n_i est le nombre de collisions du premier niveau obtenues dans la case i, *i.e.* $m_i = \#\{x \in S, h(x) = i\}$.
   Montrer que nous obtenons une table de hachage où les opérations d'insertion, de recherche et de suppression demandent un temps $O(1)$ et qui nécessite un espace de taille $O(n)$

Exercice 4. Tarbres

Un *tarbre* ou *treap* est un arbre binaire dans lequel chaque noeud a une *clé* et une *priorité*, où la suite des clés est ordonnée par ordre infixe et la priorité d'un noeud est plus petite que celle de ses fils. En d'autre terme un treap est simultanément un arbre binaire de recherche pour les clés et un tas (min) pour les priorités.



A treap. The top half of each node shows its search key and the bottom half shows its priority.

1. Montrer que la structure de l'arbre du treap est complètement déterminée par les clés et les priorités.
2. Montrer qu'un treap est l'arbre binaire de recherche qui résulte des insertions des clés dans l'ordre des priorités croissante.
3. Montrer comment réaliser les opérations et estimer leur coût en fonction de la profondeur d'un noeud (distance entre la racine et le noeud) et en fonction de n (le nombre total de noeuds) : rechercher, insérer/enlever (insérer $(S, 10)$ par exemple), un élément et séparer un treap T en deux treaps $T_<$ et $T_>$ tels que $T_<$ contient toutes les clés plus petite que la clé π et $T_>$ toutes les clés supérieures et fusionner deux treaps dont les clés de l'un sont toutes inférieures aux clés de l'autre.

   Un *treap randomisé* est un treap dans lequel les priorités sont des variables aléatoires uniformément distribuée et indépendantes continues (pour éviter des priorités égales). On va montrer que la profondeur de tout noeud est $O(\log n)$. Soit x_k le noeud qui a la k-ième plus petite clé. On définit la variable indicatrice
$$A_i^k = [x_i \text{ est un ancêtre propre de } x_k].$$
4. Exprimer la profondeur de x_k en fonction des variables indicatrices et estimer son espérance.

   On va maintenant estimer la probabilité qu'un noeud soit un ancêtre propre d'un autre. Soit $X(i,k)$ représente le sous-ensemble des noeuds $\{x_i, x_{i+1}, \ldots, x_k\}$ ou $\{x_k, x_{k+1}, \ldots, x_i\}$ selon que $i < k$ ou $k < i$.
5. Montrer que pour tout $i \neq k$, x_i est un ancêtre propre de x_k si et seulement si x_i a la plus petite priorité parmi tous les noeuds de $X(i,k)$.
6. Calculer la probabilité qu'un noeud soit un ancêtre propre d'un autre et en déduire la hauteur moyenne d'un noeud et donc le coût des opérations.
7. Pouvez-vous en déduire un nouvel algorithme de tri et montrer en quoi il ressemble à quicksort ?