

# Algorithmique et Programmation

## TD n° 4 : Hachage et tas

École normale supérieure – Département d’informatique

algoL3@di.ens.fr

2015-2016

**Quelques inégalités utiles :**

**Borne sur**  $\binom{n}{k}$

$$\frac{n^k}{k^k} \leq \binom{n}{k} \leq \frac{n^k}{k!} \leq \frac{(en)^k}{k!}$$

$$k! \geq k^{k/2}$$

**Borne union :**

$$\Pr[A \cup B] \leq \Pr[A] + \Pr[B]$$

(avec égalité si  $A$  et  $B$  sont disjoints).

**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  $\mathcal{U} \rightarrow \{0, 1, \dots, m-1\}$ .  
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 indépendamment parmi toutes les fonctions de  $\mathcal{U} \rightarrow \{0, 1, \dots, m-1\}$  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. Montrer 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 indépendamment parmi toutes les fonctions de  $\mathcal{U} \rightarrow \{0, 1, \dots, m-1\}$ . 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'élément  $y$  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 (en choisissant deux nouvelles fonctions de hachage et en reinsérant tous les éléments dans la nouvelle table). Nous supposons que  $n = \#S = m/4$ .

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  $V = \{0, \dots, m-1\}$  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 on tente d'insérer tous les éléments de  $S$ ) si et seulement si dans ce graphe il existe un ensemble de  $k$  sommets où il y a au moins  $k+1$  arêtes entre ces sommets.
3. Montrer que si on a un ensemble de  $k$  sommets avec au moins  $k+1$  arêtes entre eux dans un graphe, alors il existe un cycle dans ce graphe.  
(Un cycle est une suite de sommets  $v_1, \dots, v_\ell$  où tous les sommets consécutifs sont adjacents et  $v_1$  est adjacent à  $v_\ell$ .)
4. Montrer que la probabilité d'avoir au moins une arête entre deux sommets fixes  $v_1$  et  $v_2$  est au plus  $\frac{1}{2m}$ .  
Dans le cas où il y a bien au moins une arête, est-ce que la probabilité d'avoir une arête entre une autre paire est plus faible ou plus élevé ?  
Trouver la probabilité d'avoir une arête entre un sommet et lui-même.
5. Montrer que la probabilité que la suite  $v_1, \dots, v_\ell$  forme un cycle est au plus  $\frac{1}{(2m)^\ell}$ .
6. Montrer que la probabilité de l'événement de la question 2 est majorée par  $1/2$ .
7. En déduire que le coût amorti de l'opération d'insertion est de l'ordre de  $O(1)$  en espérance.

### Exercice 3.

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.

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  $\pi$  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  $\{x_i, x_{i+1}, \dots, x_k\}$  ou  $\{x_k, x_{k+1}, \dots, x_i\}$  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 ?