PROG
TP#7

2020-11-02

Instructions de rendu

Ce TP fera l'objet d'un rendu commun avec le TP suivant, *Tables de Hachage II*.

Tables de Hachage I : Fonction de hachage

Le but de cet exercice est d'implémenter une famille de fonction de hachage dite *polynomiale*. Une fonction de hachage est une application $\mathcal{H} : \{0,1\}^* \to \mathcal{I}$ où $\mathcal{I}$ est fixé par le contexte, qui transforme un « message » de taille variable potentiellement importante en un « haché » (ou empreinte, ou condensat) de taille fixée, généralement courte.

Une propriété importante d'une fonction de hachage est sa capacité à être « uniforme ». Informellement, on souhaite que pour toute image $y \in \mathcal{I}$, et ensemble d'entrée fini $\mathcal{S}$, $\#\{x \in \mathcal{S} : \mathcal{H}(x) = y\} \approx \#\mathcal{S}/\#\mathcal{I}$.

Le principe d'une famille de fonction de hachage polynomiale est le suivant :

1. on commence par fixer un corps fini $\mathbb{F}_q$ (qui dans notre cas sera $\mathbb{F}_{2^{33}-9} \cong \mathbb{Z}/2^{33} - 9\mathbb{Z}$) ;
2. on choisit un paramètre $k \in \mathbb{F}_q$ qui définit une instance de la famille de fonction ;
3. on découpe l'entrée m de la fonction de hachage en l blocs $m_1, \ldots, m_l$, chacun étant vu comme la représentation binaire d'un nombre se trouvant dans l'intervalle $[\![0, s]\!]$, $s \leq q$;
4. la sortie de la fonction de hachage $\mathcal{H}_k(m)$ est le résultat de l'évaluation du polynôme $M := \sum_{i=1}^{l} m_i X^i \in \mathbb{F}_q[X]$ (ou de façon équivalente $\sum_{i=1}^{l} m_i X^{l+1-i}$) en k.

Les bonnes propriétés d'uniformité d'une telle famille de fonction viennent entre autres du fait que si m et m' sont des messages découpés en au plus l blocs, $\Pr_{k\in\mathbb{F}_q}[\mathcal{H}_k(m) = \mathcal{H}_k(m')] = \Pr_{k\in\mathbb{F}_q}[\text{eval}(M - M', k) = 0] \leq l/q$, puisqu'un polynôme de degré l a au plus l racines distinctes.

Q.1 : Pour implémenter une fonction de hachage polynomiale sur $\mathbb{F}_{2^{33}-9}$, il faut tout d'abord implémenter la multiplication de deux éléments de ce corps, c'est à dire la multiplication de deux entiers modulo $2^{33} - 9$.

Écrivez une fonction `uint64_t mul339(uint64_t a, uint64_t b)` qui implémente cette multiplication. Dans un soucis d'efficacité, cette fonction ne doit pas utiliser d'instruction de division (accessible via les opérateurs `/` ou `%`).

Remarques & conseils

— $2^{33} - 9$ `= 8589934583ULL`

— $2^{32} \in [\![0, 2^{33} - 10]\!]$, mais le produit $2^{32} \times 2^{32} = 2^{64}$ provoque un *overflow* s'il est calculé avec une multiplication standard entre nombres de 64 bits. Une possibilité est donc de représenter les arguments a et b sous la forme $a_1 2^r + a_0$, $b_1 2^r + b_0$, avec un r bien choisi.

— On remarque que $2^{33} \equiv 9 \mod 2^{33} - 9$, $2^{34} \equiv 18 \mod 2^{33} - 9$, etc. Comment pouvez-vous utiliser ceci pour efficacement calculer un nombre $x < 2^{64}$ t.q. $x \equiv (a_1 2^r + a_0) \times (b_1 2^r + b_0) \mod 2^{33} - 9$?

— En utilisant la même remarque que précédemment, soit $x \in [\![0, 2^{64} - 1]\!]$, comment pouvez-vous calculer $y \in [\![0, 2 \times (2^{33} - 9) - 1]\!]$ (et de là $y' \in [\![0, 2^{33} - 10]\!]$) en effectuant la division euclidienne de x par 2^{33} ? On rappelle que votre implémentation ne doit pas avoir besoin d'utiliser l'opérateur `%`.
— Pensez à borner la taille des quantités que vous manipulez dans votre implémentation, afin de justifier sa correction.

N.B. L'algorithme suggéré est un cas particulier de la réduction de Barrett.

Testez votre fonction en la comparant sur quelques exemples avec les résultats obtenus par un calcul direct en Python.

Q.2 : Écrivez une fonction `uint64_t hash339(uint32_t k; void *buf, size_t buflen)` qui implémente une fonction de hachage polynomiale sur $\mathbb{F}_{2^{33}-9}$, où le paramètre k est pris dans $[\![0, 2^{32} - 1]\!]$, et le message est découpé en blocs de 32 bits.
— On conseille de d'abord écrire une fonction pour les messages dont la taille (en octets) est un multiple de 4, puis de traiter séparément un potentiel dernier bloc incomplet.
— Une implémentation efficace cherchera à évaluer $\sum_{i=1}^{l} m_i X^{l+1-i}$ par la « méthode de Horner », qui remarque que ce polynôme peut s'écrire $m_l X + X \times (m_{l-1} X + X \times (\ldots X \times (m_1 X) \ldots))$

Testez votre fonction en la comparant sur quelques exemples avec les résultats obtenus par un calcul direct en Python.