PROG L3-MI
ET

2021-12-06

Consignes. *La durée de cet examen terminal de deux heures. Aucun document n'est autorisé. Toute réponse doit être justifiée.*

Dans tous les exercices, il est possible d'admettre le résultat d'une question pour répondre aux questions suivantes.

Lors d'appels à des fonctions externes, faites attention à vérifier les arguments fournis et les valeurs de retour.

Exercice unique : fusion de multiensembles

On rappelle qu'un multiensemble est un ensemble pouvant contenir plusieurs fois le même élément.

On s'intéresse dans cet exercice au problème suivant : soit $\mathcal{L}_1$, $\mathcal{L}_2$ deux *multiensembles* d'entiers appartenant à l'intervalle $[\![0, 2^{32}-1]\!]$, on souhaite calculer pour $n \in [\![0, 32]\!]$ le multiensemble $\mathcal{J} := \mathcal{L}_1 \bowtie_n \mathcal{L}_2 = \{(x_1, x_2) \in \mathcal{L}_1 \times \mathcal{L}_2 : x_1 \equiv x_2 \mod 2^n\}$.

Exemple. Soit $\mathcal{L}_1 = \{1, 1, 3, 7\}$, $\mathcal{L}_2 = \{1, 15\}$, on a :
- $\mathcal{L}_1 \times \mathcal{L}_2 = \{(1,1), (1,15), (1,1), (1,15), (3,1), (3,15), (7,1), (7,15)\}$
- $\mathcal{L}_1 \bowtie_2 \mathcal{L}_2 = \{(1,1), (1,1), (3,15), (7,15)\}$
- $\mathcal{L}_1 \bowtie_3 \mathcal{L}_2 = \{(1,1), (1,1), (7,15)\}$
- $\mathcal{L}_1 \bowtie_4 \mathcal{L}_2 = \{(1,1), (1,1)\}$

Q.1 :

1. Expliquez pourquoi on peut représenter un multiensemble par la structure :

```
struct mu32
{
    uint32_t *elem;
    size_t card;
};
```

2. Écrivez une fonction `struct mu32 *alloc_mu32(size_t card)` qui alloue l'espace mémoire nécessaire pour stocker un multiensemble de `card` éléments.
3. Écrivez une fonction `void free_mu32(struct mu32 *mu)` qui libère l'espace mémoire occupé par le multiensemble fourni en argument.

Q.2 :

1. Proposez une structure `struct jmu32` permettant de stocker un produit cartésien de multiensemble.

On admettra par la suite l'existence des fonctions d'allocation et de libération mémoire associées `struct jmu32 *alloc_jmu32(size_t card)` et `void free_jmu32(struct jmu32 *jmu)`.

Q.3 :

1. Écrivez une fonction `int is_cong(uint32_t x, uint32_t y, unsigned n)` qui renvoie `1` si $\texttt{x} \equiv \texttt{y} \mod 2^{\texttt{n}}$, et `0` sinon.

Q.4 :

1. Écrivez une fonction

   ```
   void naive_join(const struct mu32 *L1, const struct mu32 *L2, struct
   ↪  jmu32 *J, unsigned n)
   ```

   qui calcule de façon naïve $\mathcal{J} := \mathcal{L}_1 \bowtie_n \mathcal{L}_2$. On supposera pour l'instant que la structure `J` a déjà été allouée de façon à toujours pouvoir stocker le résultat.
2. Quel est le coût asymptotique de votre fonction ?
3. Expliquez (sans l'implémenter) comment faire pour évaluer expérimentalement le coût de cette fonction ?
4. Quel problème pose l'allocation de la structure `J` ? Proposez (sans l'implémenter) une approche permettant de le résoudre.

Q.5 : On suppose désormais l'existence d'une fonction

```
void sort_mu32_n(struct mu32 *L, unsigned n)
```

qui trie le multiensemble `L` modulo $2^{\texttt{n}}$ (c'est à dire que deux éléments de `L` congrus entre eux modulo $2^{\texttt{n}}$ sont considérés comme égaux par la fonction de tri). Cette fonction est supposée stocker le multiensemble trié à la place du multiensemble originel (c-à-d dans la même zone mémoire).

1. Écrivez une fonction

   ```
   void join_sort1(const struct mu32 *L1, const struct mu32 *L2, struct
   ↪  jmu32 *J, unsigned n)
   ```

   qui calcule $\mathcal{J} := \mathcal{L}_1 \bowtie_n \mathcal{L}_2$ en triant au préalable *un* des multiensembles en entrée. On supposera toujours que la structure `J` a déjà été allouée de façon à toujours pouvoir stocker le résultat. Faites attention aux signatures des différentes fonctions que vous écrivez et utilisez. Pensez également à vérifier que l'algorithme utilisé renvoie bien tous les éléments recherchés.
2. Comment proposeriez vous d'implémenter `sort_mu32_n` ? (On ne vous demande pas ici d'implémenter cette fonction.)

Q.6 :

1. Écrivez une fonction

   ```
   void join_sort2(const struct mu32 *L1, const struct mu32 *L2, struct
   ↪  jmu32 *J, unsigned n)
   ```

   qui calcule $\mathcal{J} := \mathcal{L}_1 \bowtie_n \mathcal{L}_2$ en triant au préalable *les deux* multiensembles en entrée. On supposera toujours que la structure `J` a déjà été allouée de façon à toujours pouvoir stocker le résultat. Faites toujours attention aux signatures des différentes fonctions que vous écrivez et utilisez. Pensez également à encore vérifier que l'algorithme utilisé renvoie bien tous les éléments recherchés.
2. En quoi l'algorithme utilisé dans cette fonction est supérieur à celui de la fonction précédente ? Pouvez vous quantifier le gain effectué ?

N.B. Le problème étudié dans cet exercice apparaît souvent en cryptographie et en théorie des codes, notamment pour résoudre des problèmes de sac à dos ou comme sous-routine dans des algorithmes de décodage dans des codes linéaires non structurés.