Cours 6 — 3 Novembre

Enseignant : Frédéric Magniez *Rédacteur : Valentin Geffrier*

6.1 Algorithme de streaming pour les mots

6.1.1 Test d'égalité

Problème

Input: $x, y \in 0,1^n$ avec n connu, représentant des mots donnés par un stream (t, i, b)

$$\text{où} \quad \begin{cases} t = x \text{ ou } y \\ i \in \{1, \dots, n\} \\ b \in \{0, 1\} \end{cases}$$

$$\text{tels que} \quad \begin{cases} b = x_i \text{ si } t = x \\ b = y_i \text{ si } t = y \end{cases}$$

Output: ACCEPTE si $x = y$

Remarque

Nous allons utiliser la méthode FINGERPRINT vue au cours 2, grâce au théorème de Schwartz-Zippel.

Algorithme

Choisir p premier tel que $n^2 < p < 2n^2$
Choisir aléatoirement a$\in 0, \dots, p-1$
Initialiser Q_x et Q_y les polynômes associés à x et y par FINGERPRINT, au polynôme nul
Tant que le stream n'est pas vide,
- Lire le prochain (t, i, b)
- Si $t = x$, $Q_x \leftarrow Q_x + b \times a^{n-i+1}$ mod p
- Sinon, $Q_y \leftarrow Q_y + b \times a^{n-i+1}$ mod p

Si $Q_x = Q_y$, ACCEPTE
Sinon REJETTE

Théorème

Cet algorithme résout l'égalité en une passe, avec une mémoire en $O((\log n)^2 \log(\log n))$ et une erreur d'un côté inférieure ou égale à $\frac{1}{n}$.

6.1.2 PATTERN MATCHING

Problème

Input: $p \in 0,1^m$ le motif et $s \in 0,1^n$ le stream **Output:** Ensemble des i tels que p apparaît en position i de s *i.e* $p = s[i, i+m-1]$

Théorème

Il existe un algorithme qui résout le problème de PATTERN MATCHING en une passe, avec une mémoire en $O(\log(m) \times \log(n))$ et une erreur d'un côté inférieure ou égale à $\frac{1}{n}$.
Ici, nous allons construire un algorithme de mémoire $O(m \times \log(n))$

Algorithme

Choisir q premier tel que $n^3 < q < 2n^3$
Choisir aléatoirement a$\in \{0, \ldots, q-1\}$
Initialiser Q_p et Q_s les polynômes associés à p et s par FINGERPRINT, au polynôme nul
Calculer Q_p mod q en lisant les m premiers bits du stream
Calculer $Q_c(a) = Q_{s[1,p]}(a)$ mod q en lisant les p premiers bits du stream
$Q_{s[i+1,i+m]}(a) = [Q_{s[i,i+m-1]}(a) - s_i \times a^{m-1}]$ mod q $+ s_{i+m}$
$i = 1$
Tant que $i + p < n$,
- Si $Q_c = Q_p$, RENVOYER i (et continuer)
- Mettre à jour Q_c d'après la formule

$$Q_{s[i+1,i+m]}(a) = [Q_{s[i,i+m-1]}(a) - s_i \times a^{m-1}] + s_{i+m} \text{ mod q}$$

FIN

Remarque

La mémoire est bien de $O(m \log(n))$ car il faut toujours se souvenir des m derniers bits lus pour mettre à jour Q_c.

Probabilité d'erreur

Si le PATTERN apparaît en i, alors i est renvoyé ($i \in I$)
Sinon,
$\mathbb{P}(i \in I) \leq \frac{m}{n^3} \leq \frac{1}{n^2}$ car $m \leq n$
Donc $\mathbb{P}(\exists i \in I$ tel que p n'apparaît pas en $i) \leq n \times \frac{1}{n^2} = \frac{1}{n}$

6.1.3 EXPRESSION PARENTHESEE

Problème

Dyck(2): Dictionnaire $E = \{(,), [,]\}$

Input: $\omega \in E^n$ donné par le stream $\omega_1, \ldots, \omega_n$

Output: ω est bien parenthésé ou non

Cas particulier

Pour Dyck(1), $E = \{(,)\}$, il existe un algorithme déterministe de mémoire en $O(\log(n))$:
cpt=0
Tant que le stream n'est pas vide,
- Lire prochain bit s
- Si $s =$ "(", $cpt++$
- Si $s =$ ")",
- Si $cpt > 0$, $cpt--$
- Sinon STOP et RENVOYER "Erreur"

Fin
Si $cpt = 0$, RENVOYER "Ok"
Sinon, RENVOYER "Erreur"

Théorème

Il existe un algorithme qui résout le problème de Dyck(2) en une passe, avec une mémoire en $O(\sqrt{n\log(n)})$ et une erreur d'un côté inférieure ou égale à $\frac{1}{n}$.

Remarque

On peut toujours supposer que ω est bien parenthésé au type près des parenthèses et exécuter l'algorithme pour Dyck(1) en parallèle pour arrêter si l'on rencontre déjà une erreur.
On suppose donc que les cas suivants sont valides aux yeux de ce premier algorithme.

Cas 1 : Un pic d'ouverture puis de fermeture de parenthèses et de crochets de taille $2n$

Une solution déterministe est de vérifier que la suite des types de parenthèses de 1 à n est pareil de $2n$ à $n+1$.
La solution probabiliste est d'utiliser l'algorithme vérifiant l'égalité, vu en début de cours, en considérant les suites de type comme des chaînes de caractères.

Cas 2 : Plusieurs cas 1 successifs de hauteur maximale k

On utilise l'algorithme déterministe précédent avec une pile de mémoire de taille k.

Cas 3 : 2 pics successifs sans retour à une situation "nulle", c'est à dire sans parenthèses ouvertes

Choisir p premier tel que $n^3 < q < 2n^3$
Choisir aléatoirement a$\in 0, \ldots, p-1$
Initiation de la pile vide: $f_u, f_v, n_u, n_v \leftarrow 0$
Tant que le stream n'est pas vide,
- Tant que les parenthèses sont ouvrantes,
- Mettre à jour $f_u = Q_u(a)$mod p et n_u le nombre de parenthèses ouvrantes
- Tant que les parenthèses sont fermantes,
- Mettre à jour $f_v = Q_v(a)$mod p et n_v le nombre de parenthèses ouvrantes

Si $n_u = n_v$
- Si $f_u \neq f_v$, STOPPER et REJETER - Sinon, si le prochain caractère est une parenthèses fermante,
- $f_u, f_v, n_u, n_v \leftarrow 0$

Si $n_u > n_v$, $f_u, f_v, n_u, n_v \leftarrow 0$

Lemme

L'algorithme du cas 3 reconnaît Dyck(2) avec une mémoire en $O(t \log(n))$ d'erreur en $\frac{1}{n}$, où t est le nombre de pics.