TD 2: Recherche en espace constant et recombinaisons

1 Recombinaisons de mots

Exercice 1 Donner un algorithme qui teste en temps $O(n)$ si deux mots de longueurs n sont des rotations circulaires l'un de l'autre. Par exemple, *abcad* et *cadab* sont des rotations circulaires l'un de l'autre.

On considère des mots $u, v, w, \ldots$ sur un alphabet fini $\Sigma = \{a, b, \ldots\}$ et on notera $\widetilde{u}$ le mot obtenu en retournant u. P.ex. $\widetilde{abcc} = ccba$.

On dit qu'un mot u se recombine en v, noté $u \sim v$, si u peut se décomposer en $u = u_1 u_2 u_3$ de sorte que $v = u_1 \widetilde{u_2} u_3$, c.-à-d. que v est obtenu en retournant un facteur quelque part dans u. Ce genre de transformation apparaît en biologie, quand des gènes sont copiés.

Exercice 2 Montrez que $au \sim av$ ssi $u \sim v$.

Exercice 3 En déduire un algorithme efficace qui teste, étant donnés deux mots u et v, si $u \sim v$. Vous établirez sa complexité en fonction de la taille n des mots.

Soit $\mathcal{R}, \mathcal{R}'$ deux relations. La relation $\mathcal{R}\mathcal{R}'$ est définie par $u\mathcal{R}\mathcal{R}'v$ ssi $\exists w\ u\mathcal{R}w \wedge w\mathcal{R}'v$. On note $u \circlearrowleft v$ quand on peut passer de u à v par une rotation des lettres (i.e. u se décompose en xy et v en yx).

Exercice 4 Montrer que $u \sim \widetilde{v} \Rightarrow u \circlearrowleft\sim\circlearrowleft v$.

Exercice 5 Montrer que $u \sim\circlearrowleft v \Rightarrow u \circlearrowleft\sim v$ ou $u \circlearrowleft\sim \widetilde{v}$.

On étend la notion de recombinaison par la définition suivante : $u \approx v$ ssi il existe deux mots u' et v' tels que $u \circlearrowleft u' \sim v' \circlearrowleft v$ (autrement dit, $u \approx v$ ssi $u \circlearrowleft\sim\circlearrowleft v$).

Exercice 6 Proposez un algorithme efficace qui teste, étant donnés deux mots u et v, si $u \approx v$. Vous établirez sa complexité en fonction de la taille n des mots.

2 Recherche en espace constant

2.1 Recherche d'un motif auto-maximal

On fixe un ordre total sur l'alphabet, et on note $\mathit{MaxSuf}(w)$ le suffixe maximal de w au sens de l'ordre lexicographique. Le mot w est dit *auto-maximal* si $\mathit{MaxSuf}(w) = w$.

Lors du TD précédent, nous avons vu que $\mathit{Period}(w) = |w| - \pi_w(|w|)$.

Exercice 7 Montrer qu'il existe des mots qui ne sont pas auto-maximaux, peu importe l'ordre choisi sur l'alphabet.

Exercice 8 Montrer que si un mot est auto-maximal, alors tous ses préfixes le sont.

On considère l'algorithme suivant.

```
période_naïve(w,j) :=
  Period := 1;
  pour i de 2 à j faire
    si w[i] ≠ w[i − Period] alors Period := i;
  retourner Period;
```

Exercice 9 Montrer que, lorsque w est auto-maximal, période_naïve(w,j) calcule bien $Period(w[1, j])$. Pour cela, on montrera que, pour $p = Period(w[1, i - 1])$, si $w[i] \neq w[i - p]$ alors $w[i] < w[i - p]$. On montrera qu'on obtient alors une contradiction si on suppose en plus que $Period(w[1, i]) < i$.

Exercice 10 On suppose que le motif w à rechercher est auto-maximal. Adapter l'algorithme de Morris-Pratt pour le faire fonctionner en espace mémoire "constant", plus exactement, le nombre de variables codant des entiers doit rester constant (ce qui correspond en réalité à un espace logarithmique). Montrer que la complexité en temps reste linéaire.

2.2 Recherche de texte en espace constant

On suppose connue la décomposition $w = u \cdot v$ avec $v = MaxSuf(w)$, et on cherche les occurrences de w à l'intérieur d'un texte quelconque T.

Exercice 11 Si v apparaît avec un décalage de i dans T, montrer que u ne peut apparaître avec un décalage de j dans T, pour tout $j \in \{i - |u| + 1, i - |u| + 2, \ldots, i\}$.

Exercice 12 En déduire un algorithme de recherche de uv dans T en temps linéaire et espace constant.