TP 38

Algorithme de Knuth-Morris-Pratt

L'algorithme de Knuth-Morris-Pratt est un autre algorithme de recherche d'une chaîne dans un texte. Il repose lui-aussi sur un prétraitement du motif à rechercher.

1. Écrire une fonction `print_list` qui affiche une liste d'entiers formattée de la manière suivante :
   - un élément par ligne ;
   - et qui indique l'indice.

   ```
   print_list : int list -> unit
   ```

   Par exemple, l'appel `print_list [ 8; 6; 1 ]` affichera :

   ```
   indice 0 :      8
   indice 1 :      6
   indice 2 :      1
   ```

1 Méthode naïve

1. Écrire une fonction `is_prefix` tel que l'appel `is_prefix u v k` vérifie si u est préfixe du suffixe de v qui commence à l'indice k. On supposera, sans le vérifier, que la longueur de ce préfixe est supérieure à la longueur de u.

   ```
   is_prefix : string -> string -> int -> bool
   ```

2. Écrire une fonction `occurrences` tel que l'appel `occurrences u v` renvoie la liste des indices des occurrences de u dans le texte v.

   ```
   occurences : string -> string -> int list
   ```

   Par exemple :

   ```
   # print_list (occurences "abadababa" "abacabadabaabadababadababaa")
   indice 0 :      11
   indice 1 :      17
   - : unit = ()
   ```

3. Quelle est la complexité, dans le pire des cas, en fonction de $|u|$ et de $|v|$ de cette fonction ?

II Bord maximal d'un mot

Définition. Un bord d'un mot u non vide est un prefixe strict de u qui est également un suffixe de u. Le bord maximal de u est l'unique bord de longueur maximale, on le note $\beta(u)$.

Remarques.

- Tout mot non vide admet au moins le mot ε comme bord ;
- on peut identifier un bord v d'un mot u avec sa longueur $|v|$. Comme tous les bords sont préfixes de u, sa longueur suffit à caractériser un bord donné si on connaît u ;
- on pose par convention $\beta(\varepsilon) = \varepsilon$.

4. Donner les trois bords du mot $u = abaababa$. Que vaut $\beta(u)$?
5. Que vaut $\beta^{(2)}(u) = \beta(\beta((u))$? Et $\beta^{(3)}(u)$?
6. Montrer que la suite $\left(\beta^{(k)}(u)\right)_{k\in\mathbb{N}}$ converge et que l'ensemble de ses termes est exactement l'ensemble des bords de u.
7. Écrire une fonction `bord_maximal` telle que l'appel `bord_maximal u i` renvoie la longueur $\beta(u_0 \dots u_{i-1})$ du bord maximal du préfixe de taille i du mot u (supposé non vide).

   ```
   bord_maximal : string -> int -> int
   ```

   Quelle est sa complexité ?

Pour pouvoir transposer le problème de la recherche du bord maximal à la recherche d'un mot dans un texte, il nous faut calculer les bords maximaux de tous les préfixes d'un mot.

8. En utilisant la fonction précédente, écrire une fonction `bords_maximaux_prefixes_naif` qui renvoie la suite $\left(|\beta(u_0 \dots u_{i-1})|\right)_{i\in[\![0,|u|]\!]}$ (sous forme de tableau).

   ```
   bords_maximaux_prefixes_naif : string -> int array
   ```

   Par exemple :

   ```
   # bords_maximaux_prefixes_naif "abaababa";;
   - : int array = [|0; 0; 0; 1; 1; 2; 3; 2; 3|]
   ```

   Quelle est sa complexité ?

On va proposer maintenant une méthode bien plus efficace.

Supposons que l'on connaisse les bords maximaux pour les i premiers préfixes de u, c'est-à-dire jusqu'au préfixe $v = u_0u_1 \dots u_{i-1}$. On cherche alors à calculer $\beta(vu_i)$. On remarque que ce bord maximal est nécessairement de la forme wu_i avec w un bord (non nécessairement maximal) de v.

Notons $w_j = \beta^{(j)}(v)$ le j^e bord de v par taille décroissante et $k_j = |\beta^{(j)}(v)|$ sa longueur. On va donc chercher w parmi les candidats potentiels (w_j) des bords de v en commençant par le bord maximal $w_1 = \beta(v)$.

Un candidat de la forme w_ju_i est déjà suffixe de vu_i, il reste à vérifier que c'est aussi un préfixe. Comme w_j est préfixe de v, il suffit de tester si la lettre qui suit w_j dans v, celle d'indice k_j, est u_i, autrement dit se demander si $u_{k_j} = u_i$.

Si c'est le cas, $\beta(vu_i) = w_ju_i$ et c'est terminé. Sinon, on passe au bord suivant $w_{j+1} = \beta(w_j)$ de longueur $k_{j+1} = |\beta(w_j)| = |\beta(u_0u_1 \dots u_{k_j-1})|$, puisque w_j n'est autre que le préfixe de u de longueur k_j. Si on ne trouve aucun candidat qui convient, on a alors $\beta(vu_i) = \varepsilon$.

9. Dans quelle famille d'algorithmes peut-on classer ce procédé ?
10. Écrire une fonction `bords_maximaux_prefixes` qui renvoie la suite $\left(|\beta(u_0 \dots u_{i-1})|\right)_{i \in [\![0,|u|]\!]}$ (sous forme de tableau) basé sur cet algorithme.

```
bords_maximaux_prefixes : string -> int array
```

11. Montrer que la complexité de cet algorithme est linéaire en la longueur de la chaîne.

III Application à la recherche d'un mot dans un texte – Algorithme KMP

Revenons à notre problème initial de recherche d'un motif dans un texte et choisissons une lettre qui n'est pas dans l'alphabet de départ. Pour la suite, on choisira le symbole `|`.

12. Interpréter la valeur de retour de

```
bords_maximaux_prefixes (mot ^ "|" ^ texte)
```

et expliquer pourquoi cela permet de trouver les occurrences du mot dans le texte.

13. En utilisant la question précédente, écrire une fonction `occurrences_kmp` qui renvoie la liste des indices de début des occurrences d'un mot dans un texte.

```
occurences_kmp : string -> string -> int list
```

Quelle est sa complexité ?

IV Bonus – Application à d'autres problèmes

14. Un mot u est dit primitif s'il ne peut pas s'écrire sous la forme v^p avec $v \in \Sigma^*$ et $p \leqslant 2$. Écrire une fonction qui détermine en $\mathcal{O}(|u|)$ si u est primitif.
15. Déterminer en $\mathcal{O}(|u|^2)$ si un mot $u \in \sigma^*$ contient un facteur carré (c'est-à-dire un facteur de la forme ww avec w un mot non vide).
16. Écrire une fonction qui détermine en $\mathcal{O}(|u|)$ l'ensemble des préfixes de u qui sont des palindromes.