TP Option informatique $n°4$ – Nombres de palindrômes dans un mot
Le 18/01/2024

Soit Σ un ensemble fini ayant au moins deux éléments. Les éléments de Σ seront appelés des lettres. L'ensemble Σ sera appelé l'alphabet. Les suites finies d'éléments de Σ seront appelés des mots et l'ensemble des mots sera noté Σ^*.

Pour $u = u_0 \dots u_{n-1}$ un mot de Σ^*, composé de n lettres $u_i \in \Sigma, i \in \{0, 1, \dots, n-1\}$, la longueur de u sera notée $|u|$. Pour $0 \leq i < j \leq n$, on note $u[i, j]$ le mot $u_i \dots u_{j-1}$.

Pour l'implémentation en Caml, on prend pour Σ le type `char` et pour Σ^* le type `string`.

Palindromes

Définitions :

- Soit $u = u_0 \dots u_{n-1}$ un mot. On appelle miroir de u, et on note $\overline{u}$, le mot $\overline{u} = u_{n-1} \dots u_0$.
- Un mot est appelé un palindrome lorsqu'il est non vide et qu'on a $\overline{u} = u$.

1. Écrire une fonction `palindrome : string` $\rightarrow$ `bool` qui prend comme argument une chaîne de caractères `u` et retourne comme résultat un booléen indiquant si `u` est un palindrome.

2. Évaluer la complexité au pire des cas de la fonction `palindrome` en fonction de la longueur de son argument. Si c'est pire que linéaire, recommencer.

Facteurs palindromes

On recherche maintenant le nombre sous-mots de `u`, c'est-à-dire de mots de la forme `u[i,j]`, qui sont des palindromes (comptés avec les mulitplicités éventuelles).

Autrement dit, on cherche le cardinal de l'ensemble $\left\{(i, j), 0 \leq i < j \leq |u|, u[i, j] = \overline{u[i, j]}\right\}$.

Dit encore autrement, on recherche le nombre de palindromes contenus dans u.

3. Donner les 6 (comptés avec multiplicité) palindromes contenus dans le mot $u = babb$.

En parcourant naïvement les lettres d'un mot u donné, on peut proposer un algorithme en pseudo-code permettant de compter naïvement les palindromes de u (**algorithme 1**).

Algorithme 1 – Décompte naïf du nombre de palindromes contenu dans un mot donné

```
Entrees : Un mot u.
Sorties : Le nombre de palindromes contenus dans u.
Algorithme :
 nb = 0
 pour i = 0 à |u| − 1 faire
   pour j = i+1 à |u| − 1 faire
     si palindrome(u[i,j]) alors
       nb = nb + 1
     finSi
   finPour
 finPour
 retourner nb
```

4. L'implémenter. Vérifier sur *babb*.

5. Évaluer la complexité au pire des cas de l'**algorithme 1** en fonction de $|u|$.

6. Adapter l'algorithme pour qu'il retourne la liste des sous-mots palindromes de u (comptés avec multiplicité). Vérifier sur *babb*.

On souhaite bien sûr améliorer cette première idée. Pour ce faire, on utilise le paradigme de la programmation dynamique.

Pour u est un mot, on définit une matrice de booléens `m` de taille $(|u|+1) \times (|u|+1)$, `m.(i).(j)` étant vrai si `u[i,j]` est un palindrome. On particulier on a, pour tout $i \in \{0, 1, \ldots, |u|-1\}$, `m.(i).(i+1) = true`.

7. (a) À quelles conditions sur u_i, u_{j-1} et $u[i+1, j-1]$ le mot $u[i, j]$ est-il un palindrome ?
   (b) En déduire une relation de récurrence vérifiée par les coefficients de m.

8. En déduire un algorithme de programmation dynamique résolvant le problème, puis l'implémenter en CAML.

9. Adapter l'algorithme pour qu'il retourne la liste des sous-mots palindromes de u (comptés avec multiplicité).