(Inspiré de CCINP MPI 2023)

Palindromes

Un palindrome est un mot qui peut se lire indifféremment de la gauche vers la droite et de la droite vers la gauche.
En langue française, "ressasser" est le mot palindrome le plus long, tandis qu'il semble que "saippuakauppias" soit le plus long mot palindrome au monde, désignant un marchand de savon en Finlande. L'objet de ce problème est de compter le nombre de palindromes présents dans un mot donné.
Dans ce sujet, un « mot » sera repèsenté par une liste dont les éléments sont des caractères de l'alphabet : le mot "kayak" (qui est un palindrome) sera représenté par la liste `['k','a','y','a','k']`.

Définition 1 (Miroir)
Soit `L` un mot. Le *miroir* de `L = [`$c_0, \dots, c_{n-1}$`]`, noté $\overline{\texttt{L}}$, est le mot $\overline{\texttt{L}}$=`[`$c_{n-1}, \dots, c_0$`]`. Par convention $\overline{\texttt{[]}} = \texttt{[]}$.

Définition 2 (Palindrome)
Un mot `L` est un *palindrome* si et seulement si $\texttt{L} = \overline{\texttt{L}}$.

Par convention, le mot vide `[]` n'est pas considéré comme un palindrome.
On dira qu'un palindrome `L` est *pair* (respectivement *impair*) lorsque sa longueur `len(L)` est paire (resp. impaire).
On recherche donc dans ce problème le nombre de palindromes facteurs d'un mot `L` (comptés avec les multiplicités éventuelles), soit le cardinal de l'ensemble $\left\{(i,j), 0 \leqslant i < j \leqslant \texttt{len(L)}, \texttt{L[i:j]} = \overline{\texttt{L[i:j]}}\right\}$. Dit autrement, on recherche le nombre de palindromes contenus dans `L`.

1. Donner le nombre de palindromes contenus dans le mot `L = ['b','a','b','b']`.

2. Recherche exhaustive

a) Écrire une fonction `estPalindrome(L:list)->bool` qui renvoie `True` (resp. `False`) si `L` est un palindrome (resp. n'est pas un palindrome)

b) En déduire une fonction `nbPalindromes(L:list)->int` qui renvoie le nombre de palindromes facteurs de `L` en testant tous les sous-mots de `L`.

c) Évaluer la complexité de cette fonction.

3. Par programmation dynamique :

Pour un mot `L` de longueur n, on définit un « tableau » de booléens `P` de taille $(n+1)\times(n+1)$, représenté par une liste de $n+1$ sous-listes de longueurs $n+1$, tel que `P[i][j]` est vrai si `L[i:j]` est un palindrome, et faux sinon. On a donc, pour tout $i \in [\![0, n-1]\!]$, `P[i][i+1] = True`.
On rappelle que la liste `L[i:j]` est vide si $i \geqslant j$.

a) Que vaut `P[i][j]`, pour $0 \leqslant j \leqslant i \leqslant n$?

b) Justifier rapidement la relation, pour $j - i > 2$:

$$\texttt{P[i][j]} = \begin{cases} \texttt{True} & \text{si } \texttt{L[i]} = \texttt{L[j-1]} \text{ et } \texttt{P[i+1][j-1]} = \texttt{True} \\ \texttt{False} & \text{sinon} \end{cases}$$

c) En déduire une fonction `palindromes(L:list)->int` qui prend en argument un mot `L` et qui renvoie le nombre de palindromes de `L`.

d) Évaluer la complexité de cette fonction.

e) Dans cette question, on souhaite déterminer un plus long palindrome inclus dans un mot (qui n'est pas forcément unique). En remarquant qu'un plus long palindrome inclus dans une liste `L`, de longueur $n \geqslant 1$, est soit la liste `L` elle même si `L` est un palindrome, soit un plus long palindrome de la liste `L[0:n-1]`, soit un plus long palindrome de la liste `L[1:n]`.

En déduire une fonction récursive `PLP(L:list)->list` qui prend en argument une liste non vide et renvoie un plus long palindrome inclus dans `L`. On demande, même si on ne demande pas de vérifier la complexité, de rédiger une fonction de complexité au plus polynomiale (en $O(n^d)$ avec $d \in \mathbb{N}$). On pourra utiliser la fonction `estPalindrome` et introduire un dictionnaire dont les clefs sont des couples (i,j) de $[\![0,n]\!]^2$ et dont la valeur associée est un plus long palindrome de la liste `L[i:j]`.

4. Une troisième approche

On insère maintenant entre chaque paire de lettres de `L`, ainsi qu'au début et à la fin du mot, un symbole spécial noté `'#'`. On appelle ce nouveau mot $\texttt{L}^{\#}$.

Ainsi, le mot `L = ['a','b','b','a']` se transforme en $\texttt{L}^{\#}$ = `['#','a','#','b','#','b','#','a','#']`.

a) Montrer que les palindromes de $\texttt{L}^{\#}$, s'ils existent, sont tous impairs.

b) Soit `l` un palindrome de longueur k de `L`, $k \in \mathbb{N}^*$. On construit le mot $\texttt{L}^{\#}$. Donner les deux palindromes correspondant à `l` dans $\texttt{L}^{\#}$ et préciser leurs longueurs.

c) En déduire une relation entre le nombre N de palindromes de `L` et le nombre $N^{\#}$ de palindromes de $\texttt{L}^{\#}$.

On peut donc se contenter de chercher les palindromes impairs. On va donc construire un algorithme de recherche de de type de palindrome.

Définition 3 (Rayon d'un palindrome)
Soient `L` un mot de longueur n et $i \in [\![0, n-1]\!]$. On dit qu'il existe un *palindrome centré en i de rayon $\rho \geqslant 0$* si :

i) $i - \rho \geqslant 0$,
ii) $i + \rho + 1 \leqslant n$,
iii) `L[i-`ρ`:i+`ρ`+1]` est un palindrome.

Le *rayon maximal* $\hat{\rho}_i$ d'un palindrome centré en i est le plus grand rayon d'un palindrome centré en i.

Par exemple, si `L = ['a','b','b','a','b','a','b','a','b']` et $i = 4$, alors `['b']` est un palindrome centré en 4 de rayon 0, `['a','b','a']` est un palindrome centré en 4 de rayon 1, `['b','a','b','a','b']` est un palindrome centré en 4 de rayon 2 et c'est le plus grand, donc $\hat{\rho}_4 = 2$.
On peut remarquer qu'un palindrome centré en i est toujours de longueur impaire.

d) On suppose disposer d'une fonction `Manacher(L:list)->list` qui renvoie la liste `T` de sorte que `T[i]` est le rayon maximal d'un palindrome centré en i (le code de cette fonction est l'objet de la suite du sujet). En déduire une fonction `NbPalindromes(L:list)->int` qui renvoie le nombre de palindromes facteurs de `L`.

L'algorithme de Manacher, qui permet de déterminer les valeurs de $\hat{\rho}_i$, repose sur la constatation suivante : si on connaît la valeur de $\hat{\rho}_i$, alors la liste `L[`$i - \hat{\rho}_i$`:`$i + \hat{\rho}_i + 1$`]` est symétrique (par rapport à son centre i), ainsi, si $0 \leqslant j \leqslant \hat{\rho}_i$, on peut trouver un palindrome centré en $i + j$ à partir du plus long palindrome centré en $i - j$. Deux cas sont à distinguer :

- Si $\hat{\rho}_i - j \geqslant \hat{\rho}_{i-j}$, le plus long palindrome centré en $i-j$ est intégralement contenu dans la liste `L[`$i-\hat{\rho}_i$`:`$i+\hat{\rho}_i+1$`]`, donc, par symétrie, il existe un palindrome centré en $i + j$ de rayon $\hat{\rho}_{i-j}$.
- Si $\hat{\rho}_i - j < \hat{\rho}_{i-j}$, le plus long palindrome centré en $i - j$ dépasse de la liste `L[`$i - \hat{\rho}_i$`:`$i + \hat{\rho}_i + 1$`]` (à gauche), donc `L[`$i - \hat{\rho}_i$`:`$i - 2j + \hat{\rho}_i + 1$`]` est un palindrome centré en $i - j$, toujours par symétrie, il existe un palindrome centré en $i + j$ de rayon $\hat{\rho}_i - j$.

On déduit de ces deux cas, la relation $\hat{\rho}_{i+j} \geqslant \min(\hat{\rho}_i - j, \hat{\rho}_{i-j})$, si $j \leqslant \hat{\rho}_i$.
On peut donc chercher, si $0 \leqslant j \leqslant \hat{\rho}_i$, la valeur de $\hat{\rho}_{i+j}$ en commençant par l'initialiser avec la valeur $r_{i+j} = \min(\hat{\rho}_i - j, \hat{\rho}_{i-j})$ avant de chercher s'il est possible de trouver un palindrome plus grand, en testant si `L[i+j-k]` et `L[i+j+k]` sont égaux avec $k = r_{i+j} + 1, r_{i+j} + 2, \ldots$
Pour que cette recherche soit la plus efficace possible, on va introduire le *meilleur centre* qui est, parmi les centres i dont on a déjà calculé la valeur $\hat{\rho}_i$, celui pour lequel $i + \hat{\rho}_i$ est le plus grand ; cela signifie que le plus grand palindrome centré en i s'étant le plus vers la droite et augmente donc la possibilité que le plus grand palindrome centré en $i + j$ soit compris dans la liste `L[`$i - \hat{\rho}_i$`:`$i + \hat{\rho}_i + 1$`]`.
On procède donc de la façon suivante :

- On introduit une liste `T` de longueur $n =$ `len(L)` remplie de 0, qui contiendra les valeurs de $\hat{\rho}_i$, et la variable `MC`, initialisée à 0, qui contiendra l'indice correspondant au meilleur centre.
- Pour chaque entier $i \in [\![1, n-1]\!]$, afin de calculer la valeur de `T[i]`,
  - ⋆ on introduit $j = i -$ `MC`, la distance entre i et le meilleur centre `MC`.
  - ⋆ si $j \leqslant \hat{\rho}_{\texttt{MC}}$, le centre i est un des indices du plus long palindrome centré en `MC`. On initialise `T[i]` avec la valeur $\min(\hat{\rho}_{\texttt{MC}} - j, \hat{\rho}_{\texttt{MC}-j})$. Si $j > \hat{\rho}_{\texttt{MC}}$, on poursuit la recherche en partant de `T[i] = 0`.
  - ⋆ on cherche ensuite s'il est possible de trouver un palindrome plus grand en cherchant à agrandir le palindrome déjà obtenu jusqu'à trouver le rayon du plus grand palindrome centré en i.
  - ⋆ on met à jour la variable `MC` selon la valeur de $\hat{\rho}_i$ obtenue.

e) En suivant la démarche décrite au dessus, compléter le code de la fonction `Manacher(L:list)->list` de façon à ce quelle renvoie la liste `T` pour laquelle `T[i]` est la valeur de $\hat{\rho}_i$.

```
def Manacher(L) :
    n = len(L)
    T = [0 for i in range(n)]
    MC = 0
    for i in range(1,n) :
        j = i - MC
        if T[MC] >= j :
            T[i] = min(T[MC - j],T[MC] - j)
     # à compléter en rajoutant
     # autant de lignes que nécessaire
    return T
```

f) Évaluer la complexité de la fonction précédente : en notant $\texttt{MC}_i$ la valeur de la variable `MC` à la fin de la boucle d'indice i et $t_i = \texttt{MC}_i + \hat{\rho}_i$, introduire $t_i - t_{i-1}$.