Exemple de sujets - Epreuve d'Informatique, Filière MPI

Déroulement de l'épreuve

Vous disposez d'une heure de préparation pour le sujet qui suit. Une fois cette heure écoulée, vous présenterez pendant quarante-cinq minutes vos résultats devant les examinateurs qui pourront vous demander des précisions et vous poser des questions ou des exercices supplémentaires. Il est tout à fait possible de passer une question et d'en admettre le résultat pour avancer dans la préparation l'exercice. Toutefois les questions seront traitées dans l'ordre durant l'oral.

À aucun moment de l'épreuve nous n'utiliserons un ordinateur, il ne s'agit donc absolument pas d'une épreuve sur machine. Cependant, les examinateurs seront particulièrement attentifs à la capacité du candidat à écrire un programme syntaxiquement correct dans le langage `Python`. Ainsi, vous devrez être capable, pour chaque question demandant l'écriture d'un programme, de proposer un code `Python` valide.

Aucun document ni calculatrice autorisé pour cette épreuve

Palindromes et facteurs

Complexité. La complexité, ou le temps d'exécution, d'un programme P (fonction ou procédure) est le nombre d'opérations élémentaires (addition, multiplication, affectation, test, etc...) nécessaires à l'exécution de P. Lorsque cette complexité dépend de plusieurs paramètres n et m, on dira que P a une complexité en $\mathcal{O}(\phi(n,m))$, lorsqu'il existe trois constantes A, n_0 et m_0 telles que la complexité de P soit inférieure ou égale à $A \times \phi(n,m)$, pour tout $n \geqslant n_0$ et $m \geqslant m_0$.

Lorsqu'il est demandé de donner une certaine complexité, le candidat devra justifier cette dernière si elle ne se déduit pas directement de la lecture du code.

Implémentation. Dans ce sujet, nous adopterons la syntaxe du langage `Python`.

On rappelle qu'en `Python`, on dispose des opérations suivantes, qui ont toutes une complexité constante (car en `Python`, les listes sont en fait des tableaux de taille dynamique) :

- `[]` crée une liste vide (c.-à-d. ne contenant aucun élément)
- `[x]*n` qui crée une liste (ou un tableau) à `n` éléments contenant tous la valeur contenue dans `x`. Par exemple, `[1]*3` renvoie le tableau (ou la liste) `[1,1,1]` à 3 cases contenant toutes la même valeur `1`.
- `len(liste)` renvoie la longueur de la liste `liste`
- `liste[i]` désigne le (`i`$+1$)-ème élément de la liste `liste` s'il existe et produit une erreur sinon (noter que le premier élément de la liste est `liste[0]`).
- `liste.append(x)` ajoute le contenu de `x` à la fin de la liste `liste` qui s'allonge ainsi d'un élément. Par exemple, après l'exécution de la suite d'instructions "`liste = []; liste.append(2); liste.append([1,3]);`", la variable `liste` a pour valeur la liste `[2, [1, 3]]`. Si ensuite on fait l'instruction `liste[1].append([7,5]);`, la variable `liste` a pour valeur la liste `[2, [1, 3, [7,5]]]`.
- `liste.pop()` renvoie la valeur du dernier élément de la liste `liste` et l'élimine de la liste. Ainsi, après l'exécution de la suite d'instructions "`listeA = [1,[2,3]]; listeB = listeA.pop(); c = listeB.pop();`", les trois variables `listeA`, `listeB` et `c` ont pour valeurs respectives `[1]`, `[2]` et `3`.

Pour une chaîne de caractères $\texttt{s} = s_0 s_1 \cdots s_{n-1}$, on notera $|\texttt{s}| = n$ la taille de cette chaîne. Pour $i < j \in [0, n-1]$, $\texttt{s}[i:j]$ dénotera la chaîne $s_i \cdots s_{j-1}$. Toutes les chaînes obtenues de cette manière sont appelées des *facteurs* de `s`.

Pour une chaîne $\texttt{s} = s_0 s_1 \cdots s_{n-1}$, on notera $\tilde{\texttt{s}} = s_{n-1} s_{n-2} \cdots s_0$ le miroir de `s`. Par exemple, si `s = "abcd"`, son miroir $\tilde{\texttt{s}}$ est `"dcba"`.

Pour deux chaînes $\texttt{u} = u_0 u_1 \cdots u_{n-1}$ et $\texttt{v} = u_0 u_1 \cdots u_{n-1}$, on notera `uv` la concaténation $u_0 u_1 \cdots u_{n-1} v_0 v_1 \cdots v_{n-1}$ des deux chaînes.

Un *palindrome* est une chaîne qui est égale à son miroir. Par exemple, les chaînes `"kayak"` et `"leon noel"` sont des palindromes. Par contre la chaîne `"abbc"` n'en est pas un.

Question 1. Écrire une fonction `est_palindrome` qui prend en argument une chaîne de caractères et renvoie `True` si cette chaîne est un palindrome et `False` sinon. Évaluer sa complexité en fonction de la taille de la chaîne.

Question 2. Écrire une fonction `liste_palindromes` qui prend en arguments une taille `k` et une liste `L` non vide et sans doublon de chaînes de taille 1 représentant un alphabet et renvoie la liste sans doublon de tous les palindromes de taille k sur les lettres de `L`.

Par exemple, si l'on appelle `liste_palindromes(4,["a","b"])`, on obtiendra une liste contenant (dans un ordre non spécifié) `"abba"`, `"aaaa"`, `"bbbb"` et `"baab"`.

On cherche à calculer le plus grand facteur d'une chaîne qui soit un palindrome. Par exemple, le plus grand palindrome contenu comme facteur dans la chaîne `"dabbacd"` est `"abba"`.

Question 3. Écrire une fonction `palindrome_facteur_max_naif` qui prend en argument une chaîne `s` et renvoie la longueur maximale d'un facteur de `s` qui est un palindrome. Votre fonction effectuera une recherche exhaustive sur tous les facteurs de `s`. Évaluer sa complexité en fonction de $|\texttt{s}|$.

On cherche maintenant un algorithme de complexité $\mathcal{O}(|\texttt{s}|^2)$ pour résoudre ce problème.

L'idée de l'algorithme est, pour une chaîne `s` donnée en argument, de calculer une matrice M de taille $|\texttt{s}| \times |\texttt{s}|$ telle que :

$$M(i,j) = \begin{cases} \texttt{True} & \text{si } j < i, \\ \texttt{True} & \text{si } j \geqslant i \text{ et } \texttt{s}[i:j+1] \text{ est un palindrome}, \\ \texttt{False} & \text{sinon.} \end{cases}$$

L'astuce est de remplir la matrice dans un ordre judicieux de manière à n'effectuer qu'un nombre constant d'opérations pour le calcul de la valeur de chaque case.

Question 4. Écrire une fonction `palindrome_facteur_max` qui prend en argument une chaîne `s` et renvoie la longueur maximale d'un facteur de `s` qui est un palindrome. Votre fonction doit avoir une complexité en $\mathcal{O}(|\texttt{s}|^2)$.

Un *quasi-palindrome* est une chaîne de la forme $\mathtt{uv\tilde{u}}$ pour $\mathtt{u}$ une chaîne non-vide et $\mathtt{v}$ une chaîne éventuellement vide. Par exemple, `"abbefbba"` est un quasi-palindrome en prenant $\mathtt{u}$ = `"abb"` et $\mathtt{v}$ = `"ef"`. Le poids d'un quasi-palindrome $\mathtt{s}$ est la taille de la plus longue chaîne $\mathtt{u}$ telle que $\mathtt{s} = \mathtt{uv\tilde{u}}$ pour une certaine chaîne $\mathtt{v}$. Ainsi le poids de `"abbefbba"` est 3.

Question 5. Écrire une fonction `poids_quasi_palindrome` qui prend en argument une chaîne $\mathtt{s}$ et renvoie son poids si c'est un quasi-palindrome et `None` sinon.

Question 6. Écrire une fonction `quasi_palindrome_facteur_max` qui prend en argument une chaîne $\mathtt{s}$ et renvoie le poids maximal d'un facteur de $\mathtt{s}$ qui est un quasi-palindrome. La fonction renvoie `None` si la chaîne ne contient aucun facteur qui soit un quasi-palindrome. Votre fonction doit avoir une complexité en $\mathcal{O}(|\mathtt{s}|^2)$.

On oublie maintenant les quasi-palindromes et l'on revient au calcul du plus grand facteur qui est un palindrome. On cherche maintenant un algorithme en $\mathcal{O}(|s|)$.

Dans un premier temps, on ne s'intéresse qu'à la recherche de palindrome **de longueur impaire**. On verra ensuite comment traiter le cas général.

Pour $c \in [0, |\mathtt{s}| - 1]$ et $r \in [0, \min(c, |\mathtt{s}| - 1 - c)]$, le facteur de $\mathtt{s}$ centré en c et de rayon r désigne le facteur $\mathtt{s}[c - r, c + r + 1]$. Ainsi dans la chaîne `"abac"`, le facteur centré en 1 et de rayon 1 est `"aba"`.

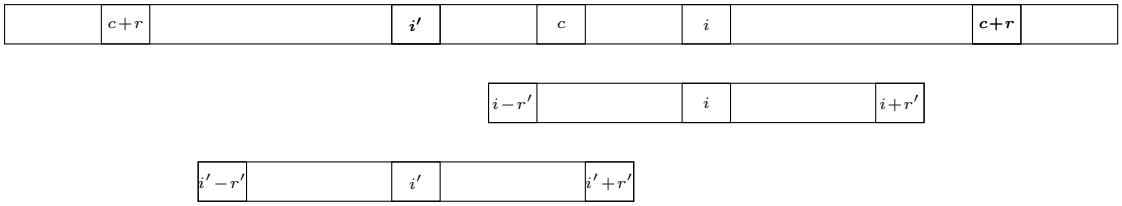


FIGURE 1 – Exemple de la situation décrite dans la Question 7.

Question 7. Soit $\mathtt{s}$ une chaîne, $c \in [0, |\mathtt{s}|-1]$ un centre et $r \in [0, \min(c, |s|-1-c)]$ un rayon tels que le facteur de $\mathtt{s}$ centré en c et de rayon r soit un palindrome. Montrer que pour tout centre $i \in [c+1, c+r]$ et pour tout rayon $r' \in [0, c+r-i]$, le facteur centré en i et de rayon r' est un palindrome si et seulement si le facteur de centre $i' = 2c - i$ et de rayon r' est un palindrome. On remarquera que i' est le symétrique de i par rapport à c.

En utilisant cette propriété, on peut écrire un algorithme en $\mathcal{O}(|\mathtt{s}|)$ qui calcule le vecteur $(M_i)_{i \in [0, |\mathtt{s}|-1]}$ tel que pour tout $i \in [0, |\mathtt{s}|-1]$, M_i est le plus grand rayon d'un facteur centré en i et qui soit un palindrome. Par exemple pour `"abababac"`, le vecteur vaudra $(0, 1, 2, 3, 2, 1, 0, 0)$.

Question 8. Écrire une fonction `palindrome_facteur_max_impair` qui prend en argument une chaîne $\mathtt{s}$ et renvoie la longueur maximale d'un facteur de taille impaire qui soit un palindrome. Votre fonction devra avoir une complexité en $\mathcal{O}(|\mathtt{s}|)$.

Pour résoudre le cas général, il suffit de considérer pour une chaîne $\mathtt{s} = s_0 s_1 \cdots s_{n-1}$, la chaîne $\overline{\mathtt{s}} = \#s_0\#s_1\#\cdots\#s_{n-1}\#$ où `#` est un symbole n'apparaissant pas dans $\mathtt{s}$. Par exemple, si $\mathtt{s}$ = `"abac"`, la chaîne $\overline{\mathtt{s}}$ sera `"#a#b#a#c#"`.

Question 9. Écrire une fonction `palindrome_facteur_max` qui prend en argument une chaîne `s` et renvoie la longueur maximale d'un facteur de `s` qui est un palindrome. Votre fonction doit avoir une complexité en $\mathcal{O}(|\texttt{s}|)$.