(PSI*)

ÉCOLE POLYTECHNIQUE

CONCOURS D'ADMISSION 2010 FILIÉRES PSI et PT

ÉPREUVE D'INFORMATIQUE

(Durée : 2 heures)

L'utilisation des calculatrices **n'est pas autorisée** pour cette épreuve.

Le langage de programmation choisi par le candidat doit être spécifié en tête de la copie. On attachera une grande importance à la concision, à la clarté, et à la précision de la rédaction.

⋆ ⋆ ⋆

Rechercher un mot dans un texte et compter ses occurrences

La quantité d'information disponible en ligne s'est considérablement accrue. Notons par exemple la numérisation électronique de nombreux ouvrages, le développement exponentiel du web ou la mise à disposition de données plus spécialisées comme le génome humain. Un problème clé lié à ce volume d'information est l'efficacité de la recherche d'informations.

Dans un moteur de recherche internet par exemple, un facteur d'importance d'une page pour une requête donnée est l'appartenance des mots recherchés à la page ainsi que leur nombre d'apparitions. Ces informations peuvent être obtenues simplement en parcourant le texte, comme nous le verrons dans une première partie. Toutefois, cette approche simple conduit à des algorithmes lents vu la taille des textes considérés. La suite du problème propose des réalisations plus efficaces.

Dans tout le problème, nous supposerons que le texte est donné dans un tableau d'entiers `tab` de taille n et que les indices de ce tableau vont de 1 à n. Chaque case du tableau contient un entier représentant une lettre. Nous supposerons donc que tous ces entiers ont des valeurs comprises entre 1 et 26. De plus, dans l'énoncé, nous noterons `tab`$[a \dots b]$, le texte extrait de `tab` composé des caractères d'indices allant de a à b, a et b compris. On utilisera indifféremment le terme caractère ou entier pour désigner le $i^{ème}$ caractère du texte ou de manière équivalente l'entier dans la case `tab`$[i]$. Dans nos exemples, pour des raisons de lisibilité, le caractère espace est utilisé mais ce caractère n'est pas encodé, seules les lettre de 'a' à 'z' sont utilisées.

Quel que soit le langage dans lequel les candidats ont choisi de composer, ils emploieront des tableaux *dynamiques*. Plus précisément, on suppose données deux primitives : une primitive `allouer`(n) qui renvoie un nouveau tableau de n cases ; et une primitive `taille`(t) qui renvoie la taille du tableau t. Par ailleurs, on suppose que les tableaux peuvent être passés en argument ou renvoyés comme résultat de fonction. Enfin, les booléens `vrai` et `faux` sont utilisés dans certaines questions de ce problème. Le candidat est libre d'utiliser les notations booléennes du langage dans lequel il compose.

Partie I. Méthode directe

Dans cette partie, nous allons mettre en œuvre des algorithmes simples permettant d'effectuer les opérations de recherche citées précédemment.

Introduisons d'abord quelques notions. Un mot est, dans notre contexte, tout simplement un texte. Un mot `mot` de taille m apparaît dans le texte `tab` de taille n, si et seulement si il existe un texte extrait de `tab`, noté `tab`$[a \dots a+m-1]$, qui est égal à `mot`, caractère pour caractère. Le *suffixe* numéro k du texte `tab` de taille n est le texte extrait `tab`$[k \dots n]$. On note que le mot `mot` apparaît dans le texte `tab`, si et seulement si il existe un suffixe tel que `mot` apparaît en tête de ce suffixe (`mot` et `tab`$[k \dots k+m-1]$ sont égaux, caractère pour caractère).

Question 1. Écrire une fonction `enTeteDeSuffixe(mot,tab,`k`)` qui renvoie `vrai` si le mot `mot` apparaît en tête du suffixe numéro k du texte `tab`, et `faux` sinon. On pourra supposer que k est un indice valide du tableau `tab`.

Question 2. Écrire une fonction `rechercherMot(mot, tab)` qui renvoie `vrai` si le mot `mot` apparaît dans le texte `tab`, et `faux` sinon.

Tester l'apparition d'un mot dans un texte ne suffit pas toujours, nombre de moteurs de recherche internet prennent en compte le nombre d'occurrences des mots recherchés dans une page donnée. Nous considérons le nombre d'occurrences avec recouvrement autorisé, qui est la notion la plus simple : on compte le nombre de répétitions du mot dans le texte, sans contrainte aucune. Par exemple, dans le texte « quelbonbonbon » (*quel bon bonbon*) le nombre d'occurrences de `bonbon` est 2, même si ces occurrences se recouvrent.

				b	**o**	**n**	**b**	**o**	**n**			
							b	**o**	**n**	**b**	**o**	**n**
q	u	e	l	b	o	n	b	o	n	b	o	n

Question 3. Écrire une fonction `compterOccurrences(mot, tab)` qui renvoie le nombre d'occurrences de `mot` dans le texte `tab`.

Nous allons maintenant calculer l'ensemble des mots d'une taille donnée qui apparaissent dans le texte ainsi que leur fréquence. Pour le moment, nous n'abordons que les tailles 1 et 2. Dans l'exemple précédent, pour la taille 1, on obtient les mots b(3), e(1), l(1), n(3), o(3), q(1), u(1), pour la taille 2, on obtient bo(3), el(1), lb(1), nb(2), on(3), qu(1), ue(1).

Question 4. Écrire une fonction `frequenceLettre(tab)` qui calcule et renvoie un tableau de taille 26 dont la case i contient la fréquence de la lettre i dans le texte.

Question 5. Écrire une fonction `afficherFrequenceBigramme(tab)` qui affiche les mots de 2 lettres présents dans le texte ainsi que leur fréquence. On suppose écrite une fonction `afficherMot(tab,`i, k`)` qui affiche le mot présent dans `tab`, commençant à l'indice i et de taille k. Le candidat proposera une réalisation simple, qui parcourt le tableau `tab` de nombreuses fois ou crée un tableau de grande taille en mémoire.

Partie II. Tableau des suffixes

Afin d'accélérer les fonctions précédentes, nous allons utiliser des algorithmes basés sur le *tableau des suffixes*, défini comme regroupant les suffixes du texte pris dans l'ordre du dictionnaire (dit ordre *lexicographique*).

Étant donné un texte `tab` de taille n, un indice k suffit à désigner un suffixe de `tab` comme le texte extrait `tab`$[k \dots n]$. Le tableau des suffixes `tabS` sera donc représenté en machine comme un tableau d'indices de `tab`. Par exemple, en prenant le texte « quelbonbonbon », on obtient les classements de la figure 1.

FIGURE 1 – Classement des suffixes Suffixes

Suffixes classés selon leur numéro

1	quelbonbonbon
2	uelbonbonbon
3	elbonbonbon
4	lbonbonbon
5	bonbonbon
6	onbonbon
7	nbonbon
8	bonbon
9	onbon
10	nbon
11	bon
12	on
13	n

Suffixes classés par ordre lexicographique

1	11	bon
2	8	bonbon
3	5	bonbonbon
4	3	elbonbonbon
5	4	lbonbonbon
6	13	n
7	10	nbon
8	7	nbonbon
9	12	on
10	9	onbon
11	6	onbonbon
12	1	quelbonbonbon
13	2	uelbonbonbon

La seconde colonne du classement de droite donne le tableau `tabS`, c'est-à-dire : [11,8,5,3,4,13,10,7,12,9,6,1,2].

Il existe des méthodes très efficaces pour calculer le tableau des suffixes, nous nous contentons d'une méthode simple.

Question 6. Écrire une fonction `comparerSuffixes`$(\texttt{tab}, k_1, k_2)$ qui prend en arguments deux suffixes du texte `tab`, représentés par leurs numéros, et renvoie un entier r. L'entier r traduit la comparaison lexicographique des *suffixes*[1] k_1 et k_2 : r est strictement négatif quand k_1 précède strictement k_2, nul quand k_1 et k_2 sont égaux, et strictement positif quand k_1 suit strictement k_2.

Question 7. Écrire une fonction `calculerSuffixes(tab)` qui prend le texte `tab` en argument et renvoie le tableau des suffixes de ce texte (tableau noté `tabS` ci-dessus). On notera que `calculer-Suffixes` effectue essentiellement un tri du tableau d'entiers $[1, 2, ..., n]$ selon l'ordre défini à la question précédente.

Partie III. Exploitation du tableau des suffixes

Nous avons déjà remarqué (Partie I) que le mot `mot` apparaît dans le texte `tab`, si et seulement si `mot` apparaît en en tête d'un suffixe de `tab`. Or le tableau `tabS` est le tableau *trié* des suffixes du texte, ce qui nous permet d'utiliser la technique de recherche *dichotomique*. Prenons l'exemple concret d'un tas t *trié* de fiches nominatives, et supposons que nous recherchions *une* fiche portant un nom donné x. On peut alors procéder ainsi :

1. S'il n'y a pas (plus, cf. ci-dessous) de fiches dans t, alors la fiche de x n'existe pas.

1. et non pas la comparaison des entiers k_1 et k_2

2. S'il y a (encore) des fiches dans t, alors on extrait une fiche située vers le milieu du tas, en prenant bien soin de distinguer le tas des fiches situées avant la fiche extraite (noté $t_{\leq}$) du tas des fiches situées après (noté $t_{\geq}$). Soit y, le nom porté sur la fiche sélectionnée.
   (a) Si y est strictement plus petit que x, alors recommencer en 1 en remplaçant t par $t_{\geq}$.
   (b) Si y est égal à x, alors on a trouvé une fiche pour x.
   (c) Si y est strictement plus grand que x, alors recommencer en 1 en remplaçant t par $t_{\leq}$.

Pour appliquer la recherche dichotomique à la recherche d'un mot dans un texte à l'aide de son tableau des suffixes, on a besoin d'une fonction de comparaison entre mot et suffixe qui élargit le cas d'égalité par rapport à une pure comparaison.

Question 8. Écrire une fonction `comparerMotSuffixe(mot,tab,`k`)` qui prend en arguments un mot `mot` et un suffixe k du texte `tab`, et qui renvoie un entier r. L'entier r traduit une légère adaptation de la comparaison lexicographique entre le mot `mot` et le suffixe k. À savoir, r est nul, si et seulement si le mot `mot` apparaît en tête du suffixe k. Autrement, r est strictement négatif (resp. positif) quand le mot précède (resp. suit) strictement le suffixe selon l'ordre lexicographique.

Question 9. Écrire une fonction `rechercherMot2(mot, tab, tabS)` qui renvoie `vrai` si le mot `mot` apparaît dans le texte `tab`, et `faux` sinon. On impose évidemment l'emploi de la technique de recherche dichotomique dans le tableau des suffixes `tabS`, que l'on suppose correct.

Question 10. Donner, sans justification, un ordre de grandeur du nombre de comparaisons de mots effectuées par `rechercherMot` (question 2) et par `rechercherMot2` (question 9). Expliquer ensuite l'intérêt du tableau des suffixes, dans le cas d'un moteur de recherche internet.

Les deux questions suivantes montrent que la technique de cette partie s'applique aussi au cas du dénombrement des apparitions d'un mot dans un texte.

Question 11. Écrire une fonction `rechercherPremierSuffixe(mot,tab,tabS)` qui renvoie le plus petit indice i de `tabS` tel que `mot` apparaît en tête du suffixe numéro `tabS[`i`]` du texte `tab`. Si `mot` n'apparaît pas dans le texte `tab`, la fonction `rechercherPremierSuffixe` doit renvoyer zéro. À titre d'exemple, dans le cas où `mot` est « bonbon » et avec le tableau des suffixes de la figure 1, `rechercherPremierSuffixe` renvoie 2. On impose évidemment une adaptation de la technique de recherche dichotomique dans le tableau des suffixes `tabS`.

On suppose écrite une fonction `rechercherDernierSuffixe(mot,tab,tabS)` analogue à la précédente, à ceci prés que `rechercherDernierSuffixe` renvoie le plus grand indice i tel que mot apparaît en tête du suffixe numéro `tabS[`i`]` du texte `tab`.

Question 12. En déduire une fonction `compterOccurrences2(mot,tab,tabS)` qui renvoie le nombre d'occurrences de `mot` dans le texte `tab`.

Nous revenons sur le calcul des sous-mots possibles abordé à la fin de la première partie et traitons la question dans toute sa généralité.

Question 13. Écrire une fonction `afficherFrequenceKgramme(tab, tabS, k)` qui affiche les mots de k lettres présents dans le texte ainsi que leur fréquence. On affichera les mots à l'aide de la fonction `afficherMot` de la question 5. Le candidat proposera une réalisation efficace qui exploite le tableau des suffixes.