Devoir surveillé : Algorithmes

L1 MPCI

2 mars 2026 - Durée: 2h

Lorsque l'on vous demande d'écrire de décrire ou de donner un algorithme cela signifiera toujours en donner un pseudo-code, justifier de son exactitude et de sa complexité

On rappelle qu'aucun document, ni équipement électrique ou électronique n'est autorisé.
Cependant *l'usage d'un marteau-piqueur à percussion sera toléré s'il est utilisé avec un silencieux.*

Les exercices de cet examen :
- sont au nombre de 3 ;
- ont tous le même but, savoir si b est une sous-chaîne de a ;
- sont indépendants (à part le problème à résoudre qui est le même) ;
- sont de difficulté croissante ;
- sont chacuns constitués de 3 parties de difficultés croissantes.

Élément de notation :
- Le nombre de points par partie est capé. **Il est impossible** d'avoir plus de 10/20 (respectivement 16/20) en ne répondant qu'aux questions des premières parties (*resp.* premières et secondes parties) de chaque exercice. **Il n'est cependant pas nécessaire** de répondre à toutes les questions pour avoir 10/20 (*resp.* 16/20) ;
- L'examen est **long** et ce qui semble simple pour l'examinateur ne l'est pas forcément pour l'étudiant et réciproquement. Il pourra être utile de changer d'exercice plutôt que de rester bloqué sur une question, quitte à y revenir plus tard.

RENDEZ DES COPIES SÉPARÉES POUR CHAQUE EXERCICE, CECI VOUS PERMETTRA DE D'Y REVENIR AU COURS DE L'EXAMEN SANS PERDRE LE CORRECTEUR.

On définit le problème algorithmique suivant :

Problème :
- **Nom :** sous-chaîne
- **Entrées :**
  - `a` une chaîne de caractères de longueur n
  - `b` une chaîne de caractères de longueur $m \leq n$
- **Sortie :** l'indice d'un commencement de b dans a si b est une sous-chaîne de a et -1 sinon

Avec la définition suivante : Soient a et b deux chaines de caractères de longueurs n et $m < n$ respectivement. La chaîne b est une ***sous-chaîne de*** a ***commençant en*** i si $b = a[i : i+m]$ (c'est à dire que pour $0 \leq i < n-m$: $b[k] = a[i+k]$ pour tout $0 \leq k < m$).

Par exemple :
- *corne*, *douille* et *oui* sont des sous-chaînes de *cornegidouille*, mais pas *rouille* ni *ubu*,
- *a*, *na* et *an* sont des sous-chaînes de *nananre*. Elles apparaissent à plusieurs endroits.

Exercice 1 – Algorithme Naïf

1.1 Algorithme python

```
def est_sous_chaîne(a, b):
    for i in range(len(a) - len(b) + 1):
        trouvé = True
        j = 0
        while j < len(b):
            if b[j] != a[i + j]:
                trouvé = False
                j = len(b)
            else:
                j += 1
        if trouvé:
            return True
    return False
```

Question 1.1.1 Donnez le pseudo-code associé à l'algorithme python ci-dessus.

Question 1.1.2 Donnez la complexité minimum et maximum de cet algorithme.

Question 1.1.3 Donnez la signature d'un algorithme résolvant le problème "sous-chaîne".

Question 1.1.4 Modifiez le pseudo-code de la question 1.1.1 pour qu'il résolve avec les mêmes complexités le problème "sous-chaîne" (il faudra bien sur démontrer son exactitude).

1.2 Bornes

Question 1.2.1 Donnez (et justifiez) une borne minimum au problème "sous-chaîne".

Question 1.2.2 Déduire de la question 1.1.4 un encadrement de la complexité du problème "sous-chaîne".

1.3 Complexité en moyenne

Question 1.3.1 En supposant l'équiprobabilité des caractères pour les différentes chaînes, montrez que $\Pr[a[i] = b[j]]$ est une constante quelques soient les indices i et j

Question 1.3.2 En déduire la complexité en moyenne de l'algorithme de la question 1.1.4 [1]

Exercice 2 – Boyer-Moore-Horspool

L'algorithme de Boyer-Moore-Horspool permet de résoudre le problème "sous-chaîne". Il fonctionne de la façon suivante :

1. on commence l'algorithme avec $i = 0$
2. pour un indice i donné, on fait décroitre un un indice j de $m - 1$ à 0.
   — si $a[i + j] = b[j]$ pour tout j, l'algorithme s'arrête et rend i
   — sinon, lors de la première non correspondance ($a[i + j] \neq b_j$ et $a[i + j'] = b[j']$ pour $j < j' < m$), l'indice i est **décalé** :
      — de m cases si $a[i + m - 1]$ n'est pas un caractère de $b[: -1]$ (la chaîne b privée du dernier caractère).
      — de $m - k - 1$ cases où k est l'indice le plus grand dans $b[: -1]$ du caractère $a[i + m - 1]$.
3. si $i > n - m$, l'algorithme s'arrête et rend -1 sinon on retourne à l'étape 2

2.1 Principe

1. Vous pourrez utiliser le fait que $\sum_{0 \leq i \leq n}(ip^i) \leq \frac{p}{(1-p)^2}$ pour tout $n \geq 0$ si $p < 1$

Question 2.1.1 Démontrez que ce fonctionnement est bien une façon valide de chercher si b est une sous-chaîne de a.

Question 2.1.2 En quoi cette amélioration permet d'aller plus vite que l'algorithme basé sur le code de la question 1.1 ?

2.2 Décalages

On suppose que les L caractères possibles (4 caractères possibles si l'on compare des brins d'ADN, 20 si ce sont des protéines et 2^{28} si ce sont des chaînes Unicode) pour les chaînes a et b sont ordonnés et que l'on possède une fonction de signature `ord(c: caractère)` $\rightarrow$ `entier` qui à tout caractère rend son ordre (0 pour le plus petit caractère et $L-1$ pour le plus grand) en $\mathcal{O}(1)$ opérations.

Question 2.2.1 Écrivez une fonction de signature `décalage(b: chaîne)` $\rightarrow$ `[entier]` qui rend un tableau T de taille L tel que $T[ord(c)]$ soit égal au décalage à effectuer pour le caractère c lors de la recherche de b dans une chaîne a.

Question 2.2.2 Quelle est la complexité de votre fonction `décalage(b: chaîne)` $\rightarrow$ `[entier]` ?

2.3 Recherche

Question 2.3.1 Écrivez le pseudo-code de l'algorithme de Boyer-Moore-Horspool.

Question 2.3.2 Quelle est la complexité maximale et minimale de l'algorithme de Boyer-Moore-Horspool ?

Question 2.3.3 Quelle est la complexité spatiale de l'algorithme de Boyer-Moore-Horspool ?

Question 2.3.4 En déduire une borne maximum au problème "sous-chaîne".

Question 2.3.5 Quand utiliseriez vous cet algorithme par rapport à l'algorithme naïf ?

Exercice 3 – Knuth-Morris-Pratt

Cet algorithme est le plus complexe des trois mais, petit *spoil*, il est optimal. Tout comme l'algorithme de Boyer-Moore-Horspool il repose sur un tableau de décalage particulier permettant d'accélérer la recherche.

```
algorithme KMP(a : chaîne, b : chaîne) → entier :
    T ←décalage(b)
    (i, j := entier) ← 0, 0
    tant que i + j < a.longueur :
        si a[i + j] == b[j] :
            j ← j + 1
            si j ≥ b.longueur :
                rendre i
        sinon :
            si j == 0 :
                i ← i + 1
            sinon :
                i ← i + j − T[j]
                j ← T[j]
    rendre −1
```

La fonction de décalage `décalage`(b : `chaîne`) $\rightarrow$ [`entier`] rend un tableau T de même longueur m que b tel que :

- T est une liste d'entier de longueur m
- $T[0] = T[1] = 0$
- pour tout $1 \leq j < m$, $T[j]$ est le plus grand entier $k < j$ tel que $b[:k] = b[j-k:j]$

Par exemple on a $[0, 0, 0, 1, 2, 3, 0]$ = décalage("$ABABACA$") :

— $T[2] = 0$ car $b[2-1:2] = "B" \neq b[:1] = "A"$
— $T[3] = 1$ car $b[3-1:3] = "A" = b[:1] = "A"$ et $b[3-2:3] = "BA" \neq b[:2] = "AB"$
— $T[4] = 2$ car $b[4-2:4] = "AB" = b[:2]$ et $b[4-3:4] = "BAB" \neq b[:3] = "ABA"$
— $T[5] = 3$ car $b[5-3:5] = "ABA" = b[:3]$ et $b[5-4:5] = "BABA" \neq b[:4] = "ABAB"$
— $T[6] = 0$ car $b[6-1] = "C"$ qui n'est pas dans $b[:5] = "A"$

3.1 Complexité Algorithme

Question 3.1.1 Montrez qu'à chaque itération soit i soit $i + j$ augmente strictement.

Question 3.1.2 En déduire que la complexité de l'algorithme de Knuth-Morris-Pratt est en $\mathcal{O}(n + f(m))$ avec :
— n et m les longueurs des chaînes a et b,
— $f(m)$ la complexité de la fonction décalage(b)

Question 3.2 Montrez l'algorithme de Knuth-Morris-Pratt permet de résoudre le problème "sous-chaîne".

3.3 Décalages

La fonction de décalage est extrêmement astucieuse :

```
fonction DÉCALAGE(b : chaîne) → [entier] :
    (T := [entier]) ← [entier]{longueur :b.longueur}
    T[0], T[1] ← 0, 0
    (i, j, k := entier) ← 2, 2, 0
    (c := caractère) ← b[j − 1]
    tant que i < T.longueur :
        k ← T[j − 1]
        si c == b[k] :
            T[i] ← k + 1
            i ← i + 1
            j ← i
            c ← b[j − 1]
        sinon :
            si k ≤ 1 :
                T[i] ← 0
                i ← i + 1
                j ← i
                c ← b[j − 1]
            sinon :
                j ← k + 1
    rendre T
```

Question 3.3.1 Montrez qu'à chaque itération soit :
— k reste constant à 0
— k augmente de 1
— k diminue strictement

Question 3.3.2 En déduire que la complexité de la fonction `décalage`(b) est en $\mathcal{O}(m)$ avec m la taille de la chaîne passée en paramètre de la fonction.

Question 3.3.3 Montrez que la fonction décalage correspond bien à celle demandée pour faire fonctionner l'algorithme de Knuth-Morris-Pratt.

Exercice 4 – Exercice secret final et bonus

Déduire de tout ce qui précède la complexité de problème "sous-chaîne".