Recherche textuelle EXERCICES

Dernière mise à jour le : 06/05/2024

Exercice 1 : Recherche naïve

À la main

On considère le motif M = `ar` à cherche dans le texte T = `barbara`.

✍ **Question 1** : Faites un schéma en faisant glisser le motif le long du texte suivant l'algorithme de recherche naïve. On attend l'état à chaque étape du glissement et pour chaque étape, le nombre de comparaisons nécessaires. Combien de comparaisons sont nécessaires en tout.

✍ **Question 2** : Quelles sont les positions en lesquelles le motif a été trouvé ?

✍ **Question 3** : Pour un texte de taille n et un motif de taille m, quelle est la dernière position à tester pour chercher le motif ? *Un schéma n'est pas inutile.*

Coût de l'algorithme naïf

✍ **Question 4** : Pour le texte T = `AAAAAAAAA` :

- donnez un motif de taille 3 qui serait un *pire cas* (en nombre de comparaisons) ;
- donnez un motif de taille 3 qui serait un *meilleur cas* (en nombre de comparaisons)

✍ **Question 5** : En déduire la complexité temporelle de la recherche naïve pour un texte T de taille n et un motif M de taille m.

Implémentation de l'algorithme naïf

L'objectif de cette dernière partie est d'implémenter une fonction `recherche_naive(M, T)` qui renvoie une liste contenant toutes les positions du motif `M` dans le texte `T`, obtenues en utilisant l'algorithme de recherche naïve.

Cette fonction utilisera une fonction auxiliaire `correspond(M, T, i)` qui renvoie `True` si on trouve `M` dans `T` exactement en position `i`, et `False` sinon.

🐍 **Question 6** : Écrivez les deux fonctions en question et vérifiez que les tests proposés soient validés. *Contrainte : en une position donnée, la recherche se fera avec une boucle* `while`.

> On rappelle que l'on notera toujours `i` la position dans `T` et `j` la position dans `M`.

```
# à vous de jouer !
def correspond(M, T, i):
    pass

def recherche_naive(M, T):
    pass

# tests
assert correspond('GCAG', 'GGCAGCCGAACCGCAGCAGCAC', 1) == True
assert correspond('GCAG', 'GGCAGCCGAACCGCAGCAGCAC', 0) == False
assert correspond('BRA', 'ABRACADABRA', 8) == True

assert recherche_naive('GCAG', 'GGCAGCCGAACCGCAGCAGCAC') == [1, 12, 15]
```

```
assert recherche_naive('TOTO', 'GGCAGCCGAACCGCAGCAGCAC') == []
assert recherche_naive('ar', 'barbara') == [1, 4]
```

Exercice 2 : Algorithme de Boyer-Moore-Horspool (BMH)

À la main

On considère le motif M = `abracadabra` à cherche dans le texte T = `bradacadabdabracadabratabracadabra`.

✍ **Question 1** : Complétez le schéma en faisant glisser le motif le long du texte suivant l'algorithme de Boyer-Moore-Horspool. On attend l'état à chaque étape du glissement et pour chaque étape, le nombre de comparaisons nécessaires. Combien de comparaisons sont nécessaires en tout. (*Ne vous en faites pas, il n'y a que 6 positions à tester*)

```
b r a d a c a d a b d a b r a c a d a b r a t a b r a c a d a b r a
```

✍ **Question 2** : Quelles sont les positions en lesquelles le motif a été trouvé ?

> **Remarque** : comme pour l'algorithme naïf, pour un texte de taille n et un motif de taille m, la dernière position à tester est $n - m$.

Efficacité de l'algorithme de BMH

✍ **Question 3** : Pour le texte T = `AAAAAAAAAA` :

- donnez un motif de taille 3 qui serait un *pire cas* (en nombre de comparaisons) ;
- donnez un motif de taille 3 qui serait un *meilleur cas* (en nombre de comparaisons)

Prétraitement du motif : construction de la table du mauvais caractère

✍ **Question 4** : Donnez le dictionnaire correspondant à la règle du mauvais caractère pour le motif M = abracadabra (voir cours si nécessaire)

🐍 **Question 5** : Écrivez une fonction `table_MC` qui prend en paramètre une chaîne de caractères `M` et qui renvoie le dictionnaire correspondant à la règle du mauvais caractère du motif `M`, à savoir un dictionnaire qui associe à chaque caractère du motif (excepté le dernier) le nombre de positions à "remonter" à partir de la fin du motif pour le rencontrer (on cherche donc la distance entre la dernière occurrence de chaque caractère et la fin du motif).

Exemples :

```
>>> table_MC('GCAG')
{'G': 3, 'C': 2, 'A': 1}
```

```
>>> table_MC('SPECIALITE')
{'S': 9, 'P': 8, 'E': 7, 'C': 6, 'I': 2, 'A': 4, 'L': 3, 'T': 1}
```

Question 6 : Vérifiez en faisant l'appel `table_MC("abracadabra")`.

Implémentation de l'algorithme de BHM

Question 7 : Complétez la fonction `BHM` qui renvoie une liste des positions du motif `M` dans le texte `T` en appliquant l'algorithme de BMH.

```
# À vous de jouer !

def BMH(M, T):
    # Prétraitement (règle du mauvais caractère)
    MC = table_MC(M)

    # Recherche par fenêtre glissante
    n = len(T)
    m = len(M)
    positions = []
    i = 0
    while i <= ... - ...:  # i : position du motif
        j = ...  # comparaison à partir de la droite !
        while j >= ... and M[...] == T[i + ...]:
            j = ...
        if ...:  # si motif trouvé
            positions.append(i)
            i = i + 1  # décalage de 1
        else:  # si motif non trouvé --> caractère fautif : T[i+j]
            if T[i+j] in MC:  # si le caractère fautif T[i+j] est dans le motif
                i = i + ...  # on fait le décalage en fonction de MC (DIFFICILE)
            else:  # si le caractère fautif T[i+j] n'est pas dans le motif
                i = ...  # on place le motif juste après la position fautive
    return positions

# Tests
assert BMH('GCAG', 'GGCAGCCGAACCGCAGCAGCAC') == [1, 12, 15]
assert BMH('TOTO', 'GGCAGCCGAACCGCAGCAGCAC') == []
assert BMH('ar', 'barbara') == [1, 4]
```

Efficacité en pratique

Le fichier ltdme80j.txt contient le texte de l'oeuvre *Le tour du monde en 80 jours* de Jules Verne. On peut le charger dans la chaîne de caractères `texte` (de longueur 428969) avec le code suivant :

```
fichier = open("ltdme80j.txt", mode = "r", encoding = "utf-8")
texte = fichier.read()  # mémorisation dans une unique chaîne de caractères
fichier.close()
print(len(texte))
print(texte)
```

Question 8 : Comparez le temps mis par la recherche naïve et la recherche selon l'algorithme de Boyer-Moore-Horspool si on cherche le mot `"Phileas"` (prénom du personnage principal de l'oeuvre).

Références

- Documents ressources du DIU EIL, Université de Nantes.

Germain BECKER, Lycée Mounier, ANGERS

Voir en ligne : info-mounier.fr/terminale_nsi/algorithmique/recherche-textuelle-exercices