9.- Algorithmique du texte. Exemples et applications

Niveau : MP2I / SI

Prérequis : Analyse d'algorithmes + Programmation C

Motivation : Le traitement de textes est inhérent à l'informatique. L'algorithmique du texte permet un gain en efficacité important: elle est à la base des fonctions courantes comme Rechercher/Remplacer, et en bio-informatique.

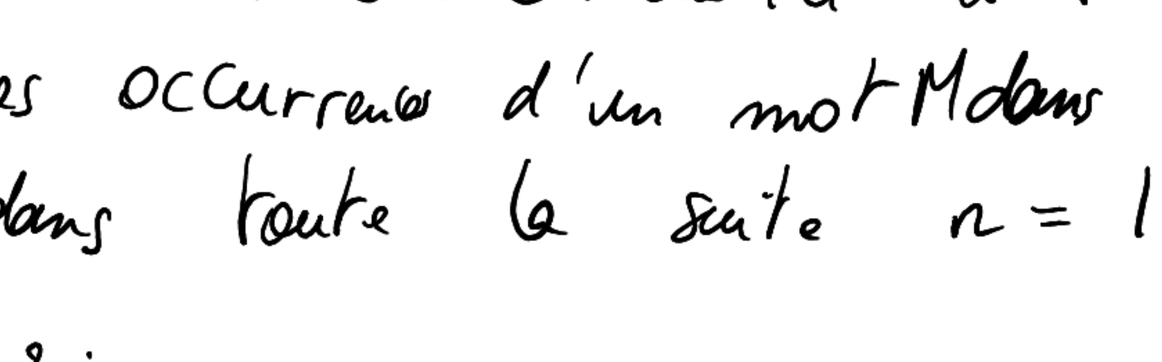
I] Recherche d'un motif dans un texte

1.) Introduction au problème

Proposition 1 : On cherche ici à trouver efficacement toutes les occurrences d'un mot M dans un texte T. On notera dans toute la suite $n = |T|$ et $m = |M|$.

Problème 2 :
- Entrées : Mot $M \in \Sigma^*$, texte $T \in \Sigma^*$.
- Sortie : Ensemble des indices i tels que $T[i : i+m] = M$.

Exemple 3 : Pour M = bra et T = abracadabra, on a :



Il y a donc 2 occurrences de M dans T, aux positions 1 et 8.

2.) Algorithme naïf :

Idée 4: Pour chaque indice i, comparer les chaînes de caractères $T[i : i+m]$ et M, et ajouter i à l'ensemble des indices valides en cas de réussite.

Complexité 5: Cet algorithme effectue au plus $m \times (n-m+1)$ comparaisons. Toutes les opérations s'effectuent en temps constant, donc cet algorithme a une complexité de $O(n \times m)$.

3.) Algorithme de Boyer - Moore

Idée 6: Tester tous les décalages de M en commençant les comparaisons à la droite du motif. Si tous les caractères coïncident, on a bien une occurrence, sinon soit j l'indice de la première différence, i.e le plus grand entier tel que $j \in [\![0; m-1]\!]$ et $M[j] \neq T[i+j]$: on augmente l'indice i d'un certain nombre dépendant de M pour éviter les calculs inutiles. Précisément, on augmente i de $j-k$ où $k = \underset{k \in [\![0; j-1]\!]}{\operatorname{argmax}} (M[k] = T[i+j])$ s'il existe, ou $j+1$ sinon

Idée 7: Plutôt que recalculer k à chaque étape, on peut précalculer une table des décalages pour le motif donné.

Exemple 8: Table de décalage pour le motif "abracadabra"

Pour une position $j \in [\![0; m-1]\!]$ et un caractère c, la table donne le $k \in [\![0; j-1]\!]$ maximal, s'il existe, tel que $M[k] = c$.

	0	1	2	3	4	5	6	7	8	9	10
a		0	0	0	3	3	5	5	7	7	7
b			1	1	1	1	1	1	1	8	8
r				2	2	2	2	2	2	2	9
c						4	4	4	4	4	4
d								6	6	6	6

Activité 9 : Écrire l'algorithme de Boyer-Moore en C.

Complexité 6 : La construction de la table s'effectue en $O(m^2)$. Dans le pire cas, le coût de la recherche s'effectue en $O(m*n)$. En revanche, dans le meilleur cas, la recherche s'effectue en $O(\frac{n}{m})$.

Remarque 11 : On peut améliorer l'algorithme avec d'autres tables de décalage pour avoir une complexité dans le pire cas en $O(m+n)$.

4.) Algorithme de Robin-Karp

Idée 12 : Précalculer un hachage de la chaîne de caractère à tester avant d'effectuer la vraie comparaison : on calcule le hachage de M et celui de $T[i : i+m]$ à chaque position i, et si les hachages sont distincts alors on est certain que $M \neq T[i : i+m]$.

Idée 13 : Calculer le hachage de manière "glissante" : on réutilise une grande partie du calcul effectué à l'indice $i-1$ pour calculer le hachage de l'indice i.

Exemple 13: Pour $\Sigma = \{a, b, c\}$, on utilise la fonction de coût v : $a \mapsto 0$, $b \mapsto 1$, $c \mapsto 2$, et on a alors la fonction de hachage suivante : $h(c_0 \cdots c_{m-1}) = \sum_{i=0}^{m-1} B^{m-1-i} v(c_i)$, avec B une constante. On a alors :
$h(c_{i+1} \cdots c_{i+m}) = B \times \left(h(c_i \cdots c_{i+m-1}) - B^{m-1} c_i\right) + c_{i+m}$.

Complexité 14: Dans le meilleur cas, cet algorithme a une complexité de $O(n+m)$. Dans le pire cas en revanche il a une complexité de $O(n \times m)$ (comparaison à chaque indice).

5.) Recherche d'un motif par un automate fini

DEV : Recherche d'un motif par un automate fini

II] Compression :

1.) Introduction aux algorithmes de compression

Définition 15: On appelle compression le fait de réduire l'espace occupé par une information.

Exemples 16: On utilise très régulièrement de la compression dans des formats d'image (JPEG), de musique (MP3) ou encore de vidéo (MP4).

Remarque 17: On s'intéresse ici à la compression d'un texte.

Définition 18: On appelle taux de compression le rapport entre l'espace occupé par l'information d'origine et l'information compressée.

2.) Algorithme de Huffman

DEV : Algorithme de Huffman

3.) Algorithme de Lempel-Ziv-Welch

a.- Compression

Idée 19 : Rechercher dans le texte à compresser des répétitions de sous-chaînes identiques, et à leur donner une forme compacte dans le texte compressé.

Remarque 20 : Contrairement à l'algorithme de Huffman, l'algorithme LZW procède en une seule passe : il permet de compresser le texte d'entrée au fur et à mesure de sa lecture. En cela, il est adapté à la compression d'un document trop gros pour être lu entièrement avant compression.

Idée 21 : On maintient un dictionnaire qui associe à des sous-chaînes du texte à compresser des codes qui les représentent.

Remarque 22 : Contrairement à l'algorithme de Huffman, l'algorithme LZW ne fait pas correspondre un caractère du texte initial avec un code du texte compressé : plusieurs caractères à la suite peuvent être encodés par un seul code.

Exemple 23 : Exemple minimal de compression par l'algorithme LZW
Texte initial : "ENTENDENT", $\Sigma = \{$"E", "N", "T", "D"$\}$.
Dictionnaire initial :
· E ↦ 0, N ↦ 1, T ↦ 2, D ↦ 3.

Nom : Sujet : Jury : Page : /

Entrée ajoutée	Texte restant	Résultat
E↦0, N↦1, T↦2, D↦3	ENTENDENT	
EN↦4	NTENDENT	0
NT↦5	TENDENT	01
TE↦6	ENDENT	012
END↦7	DENT	0124
...	...	...

b.- Décompression

Idée 24: On souhaite ici retrouver le texte initial à partir du texte encodé résultant de la phase de compression. Pour cela, on a seulement besoin des entrées initiales du dictionnaire, ce qui est très peu coûteux à stocker en terme d'espace.

Exemple 25: On reprend le même exemple.

Entrée ajoutée	Texte restant	Résultat
E↦0, N↦1, T↦2, D↦3	01243 42	
	1243 42	E
EN↦4	243 2	EN
NT↦5	43 2	ENT
...	...	...

Activité 26: Programmer en C ces deux algorithmes et comparer les taux de compression obtenus sur un long exemple.