Algorithmique du texte

Annie Chateau

Plan

Introduction

Le texte est l'une des représentations de l'information la plus simple et naturelle.

Les données à traiter se présentent souvent comme une suite de caractères (fichiers textes par exemple).

Les *textes* sont l'objet central du traitement de texte sous toutes ses formes.

Nous allons en voir quelques aspects dans ce cours. . .

Introduction

L'algorithmique du texte a des champs d'application dans la plupart des sciences et à chaque fois qu'il y a du traitement de l'information à réaliser.

La plupart des éditeurs de texte et des langages de programmation proposent des outils pour manipuler les textes (ou chaînes de caractères).

La complexité des algorithmes de traitement de texte est un domaine de recherche très actif, où la théorie et la pratique sont très proches!

Le *pattern matching*

Le problème le plus fondamental en traitement de texte est le problème du *pattern matching*, ou recherche de motif dans un texte.

Il est nécessaire pour accéder à l'information.

Il est comparable, en terme d'utilité, aux tris sur les structures de données, ou aux opérations arithmétiques. . .

Le *pattern matching*

L'expression la plus simple du problème : on recherche une chaîne de caractères de longueur m, le motif (*pattern*) dans une autre chaîne de caractères de longueur $n > m$, le texte.

Le motif peut être décrit simplement (motif exact) ou de manière à désigner plusieurs chaînes de caractères (motif approché).

La longueur du motif et celle du texte peuvent être très très grandes $\Rightarrow$ il est impératif de considérer la complexité des algorithmes, qui sera la plupart du temps polynomial, avec un degré le plus faible possible...

Manipulation de fichiers textes

Le système Unix utilise les fichiers textes pour échanger l'information. L'utilisateur peut extraire des informations des fichiers et les transformer à travers un certain nombre de commandes.

Les outils sont en général des *filtres* qui lisent une entrée et produise une sortie simultanément.

On peut les connecter les uns aux autres (pipeline).

Exemple

La commande `grep` (general regular expression) répond au problème de la recherche de motif, et également à celui de la recherche d'*expression régulière* (regexp).

Manipulation de fichiers textes

L'outil `sed` (stream editor) peut remplacer des motifs par des chaînes de caractères spécifiées. C'est un outil très puissant.

Exemple

On peut, avec `sed`, avoir une action sur les lignes contenant deux fois un mot particulier, ou supprimer deux occurrences consécutives du même mot dans un texte...

Cet outil répond aux problèmes de recherche de répétitions, de recherche de motifs ou plus généralement de la recherche de régularité dans une chaîne de caractères.

Dictionnaires

La recherche de mots ou de motifs dans des structures statiques est un problème légèrement différent du problème précédent.

Dans un dictionnaire, les entrées sont classées par ordre alphabétique pour accélérer la recherche d'une entrée.

On retrouve la classe de problèmes relative aux tris!

Dans un ouvrage technique, par exemple un livre d'algorithmique du texte, il y a également un **index**, constitué de termes choisis, qui donne des pointeurs vers les parties du textes relatives à ces termes.

Les problèmes relatifs à la création d'index forment un groupe à part...

Compression de données

Un des problèmes essentiels dans la gestion de données de grande taille est la compression de texte.

Il s'agit de changer de représentation afin de réduire la taille prise par cette représentation, en assurant que l'on puisse retrouver les données de départ $\Rightarrow$ sans perte d'information, contrairement à ce qui peut se produire pour compresser des images ou du son.

La compression de texte sans perte se fonde grandement sur la recherche de régularité dans un texte.

Plan

Mot

Definition (Alphabet)

Un alphabet Σ est un ensemble, non vide, fini ou infini de *symboles*.

Exemple

Pour $k \geq 2$, l'alphabet $\Sigma_k = \{0, 1, \ldots, k-1\}$.

Definition (Mot)

Un mot (ou chaîne de caractères) de l'alphabet Σ est une liste de symboles issus de Σ.

Exemple

3425 est un mot de l'alphabet Σ_6.

Mot

Remarques :

1. un mot peut être fini ou infini
2. un mot fini de longueur n peut être vu comme une application de $\{0, \ldots, n-1\}$ vers Σ.
3. un mot de longueur $n = 0$ est le mot *vide*, noté ϵ.

Definition (Ensemble des mots d'un alphabet)

L'ensemble des mots finis d'un alphabet Σ est noté Σ^*.
L'ensemble des mots finis *non vides* de l'alphabet Σ est noté Σ^+.

Mot

Exemple

Si $\Sigma = \{a, b\}$, alors $\Sigma^* = \{\epsilon, a, b, aa, ab, bb, aaa, aab, \ldots\}$.

Definition (Longueur d'un mot)

Si w est un mot fini, sa *longueur* (i.e. le nombre de symboles contenus dans w) est notée $|w|$.

Exemple

Le mot `cinq` est de longueur 4.

Remarque : $|\epsilon| = 0$**.**

Mot

Definition (Occurrence d'un symbole)

Si $a \in \Sigma$ et $w \in \Sigma^*$, alors $|w|_a$ désigne le nombre d'occurrences du symbole a dans le mot w.

Exemple

$|\texttt{occurrence}|_{\texttt{r}} = 2$
$|\texttt{occurrence}|_{\texttt{c}} = 3\ldots$

Concaténation

Definition (Concaténation de deux mots)

La concaténation de deux mots finis w et x est la juxtaposition des symboles de w et des symboles de x, notée wx.

Exemple

Si $w = \texttt{titi}$ et $x = \texttt{tata}$ alors $wx = \texttt{tititata}$.

Remarque : La concaténation n'est pas *commutative* $wx \neq xw$, mais elle est *associative* : $(xy)z = x(yz)$.

Concaténation

Remarque : la concaténation est notée comme la multiplication, c'est-à-dire que w^n désigne $w \dots w$ (n fois).

L'ensemble Σ^* muni de la loi de composition interne *concaténation* est un monoïde, avec comme élément neutre le mot vide ϵ.

Facteur

Definition (Facteur)

On dit qu'un mot y est un *facteur* d'un mot w s'il existe des mots x et z tels que $w = xyz$.
Le mot x est un *préfixe* (resp. propre) du mot w s'il existe un mot y (resp. $\neq \epsilon$) tel que $w = xy$.
Le mot z est un *suffixe* (resp. propre) du mot w s'il existe un mot y (resp. $\neq \epsilon$) tel que $w = yz$.

Exemple

On considère $w = \texttt{barbapapa}$. Le mot $x = \texttt{bar}$ est un préfixe de w, $y = \texttt{papa}$ est un suffixe de w, et $y = \textit{rbapa}$ est un facteur de w.

Sous-mot

Si $w = a_0 a_1 \dots a_{n-1}$ alors pour $i \in \{0, \dots n-1\}$, on définit : $w[i] = a_i$.

Si $i \in \{0, \dots n-1\}$ et $i - 1 \leq j \leq n-1$, on définit : $w[i \dots j] = a_i \dots a_j$.

Remarque : $w[i \dots i] = a_i$ et $w[i \dots i-1] = \epsilon$.

Palindromes

Definition (Mot miroir)

Si $w = a_0 a_1 \dots a_{n-1}$ est un mot fini de Σ, on appelle *mot miroir* de w, noté $\overline{w}$, le mot $a_{n-1} a_{n-2} \dots a_0$.

Exemple

Le mot `siort` est le mot miroir du mot `trois`.

Definition (Palindrome)

On appelle *mot palindrome* un mot w fini qui est identique à son mot miroir.

Exemple

Le mot `kayak` est un palindrome.

Période

Definition (Période)

Une *période* d'un mot w est un entier $0 < p \leq |w|$ tel que

$$\forall i \in \{0, \ldots, |x| - p - 1\}\, x[i] = x[i + p].$$

On note $period(x)$ la plus petite période de x.

Exemple

Les périodes de aabaaabaa (de longueur 9) sont 4, 7, 8 et 9.

Période

Proposition

Soit x un mot non vide et p un entier tel que $0 < p \leq |x|$. Chacune des conditions équivalentes suivantes définit une période :

1. *x est un facteur d'un mot y^k avec $|y| = p$ et $k > 0$,*
2. *x peut être écrit sous la forme $(uv)^k u$ avec $|uv| = p$, v non vide et $k > 0$,*
3. *il existe des mots y, z et w tels que $x = yw = wz$ et $|y| = |z| = p$.*

Bord

Definition (Bord)

Un bord du mot x est un sous-mot de x qui est à la fois un préfixe et un suffixe de x.

Exemple

Les bords du mot aabaaabaa sont aabaa, aa, a et ϵ.

Remarque : bord et période sont des notions duales.

Bord

Definition (Bord maximal)

Soit x un mot non vide. On note $Border(x)$ le plus grand bord propre de x (*i.e.* différent de x). On dit que x est *sans bord* si $Border(x) = \epsilon$.

Remarque : Le bord d'un mot est le plus long chevauchement non trivial quand on essaye de faire coïncider x avec lui-même. Puisque $Border(x)$ est strictement plus petit que x, si on itère le processus on finit par tomber sur le mot vide ϵ.

Bord

Proposition

Soit x un mot et $k \geq 0$ le plus petit entier tel que $Border^k(x)$ est vide. Alors

$$(x, Border(x), Border^2(x), \ldots, Border^k(x))$$

est la suite de tous les bords de x ordonnés par longueur décroissante, et

$$(|x| - |Border(x)|, |x| - |Border^2(x)|, \ldots, |x| - |Border^k(x)|)$$

est l'ensemble de toutes les périodes de x en ordre croissant.

Remarque : On a exactement $period(x) = |x| - |Border(x)|$.

Bord

Les résultats suivants sont utilisés dans les preuves combinatoires sur les mots.

Lemma (Périodicité faible)

Soient p et q deux périodes d'un mot x. Si $p + q \leq |x|$, alors $\mathrm{pgcd}(p, q)$ est aussi une période de x.

Lemma (Périodicité)

Soient p et q deux périodes d'un mot x. Si $p + q - \mathrm{pgcd}(p, q) \leq |x|$, alors $\mathrm{pgcd}(p, q)$ est aussi une période de x.

Mots de Fibonacci

Dans un sens, cette inégalité est optimale. Le but de l'exemple suivant est de trouver un mot x qui possède deux périodes p et q telles que $p + q - \mathrm{pgcd}(p, q) = |x| + 1$ et ne satisfaisant pas la conclusion du lemme.

On considère l'alphabet $\Sigma = \{a, b\}$. On définit les mots de Fibonacci par :

$$\begin{cases} Fib_0 &= \epsilon \\ Fib_1 &= b \\ Fib_2 &= a \\ Fib_n &= Fib_{n-1}Fib_{n-2} \text{ pour tout } n \geq 2 \end{cases}$$

Mots de Fibonacci

Les longueurs des mots de Fibonacci sont exactement les termes de la suite de Fibonacci...

On définit g_n le préfixe de Fib_n de longueur $|Fib_n| - 2$. Alors g_n vérifie les conditions requises pour prouver l'optimalité du lemme de périodicité...

Plan

Algorithme de recherche naïf

On souhaite rechercher toutes les occurrences d'une chaine de caractères `motif` de longueur m dans une chaine de caractères `texte` de longueur n. On supposera $m \leq n$.

Algorithme de recherche naïf

On souhaite rechercher toutes les occurrences d'une chaine de caractères `motif` de longueur m dans une chaine de caractères `texte` de longueur n. On supposera $m \leq n$.

Exemple : Rechercher "de" dans le texte précédent renverra les quatre positions 59, 79, 109, 129.

Algorithme de recherche naïf

Entrées: String motif, string texte
Sorties: Liste des positions du motif (premier caractère)
fonction *Recherche_motif(motif,texte)*

```
position_motif=[]
n=longueur(texte)
m=longueur(motif)
pour i allant de 0 à n-m-1 faire
    j=0
    tant que j<m et texte[i+j]==motif[j] faire
        j=j+1
    si j==m alors
        /*Alors texte[i+j]==motif[j] a toujours été vraie*/
        Ajouter i à position_motif
retourner position_motif
```

Algorithme de recherche naïf

```
1  def rech_motif(mot,t):
2
3      liste_pos=[]
4      n=len(t)
5      m=len(mot)
6      for i in range(n-m):
7          j=0
8          while(j<m and t[i+j]==mot[j]):
9              j+=1
10         if j==m:
11             liste_pos.append(i)
12     return liste_pos
```

Algorithme de recherche naïf

- Unité de mesure : les comparaisons
- Boucle `for` : une boucle `while` $n - m$ fois et $n - m$ comparaisons (embranchement)
- Boucle `while` : au pire on trouve à chaque fois le motif $\Rightarrow 2m$ comparaisons
- Complexité au pire : $\mathcal{O}(m(n - m))$.