Vérification de l'accès à un site WEB

Patrick CEGIELSKI
Université Paris 12,
IUT de Fontainebleau
Route forestière Hurtault,
F-77300 Fontainebleau,
cegielski@univ-paris12.fr

Résumé

Nous montrons comment un problème pratique lié à la consultation des pages Web est modélisé en un problème algorithmique et comment trouver un algorithme efficace pour celui-ci.

Notre but n'est pas ici tant de répondre à une question précise que de montrer comment, sur un exemple concret précis, l'informaticien conçoit des modèles, dans le même sens que les physiciens, conduisant à des problèmes mathématiques (plus particulièrement des questions d'informatique théorique) dont la solution est loin d'être immédiate.

Mots clés : Web, algorithme, complexité.

1 Un problème pratique

Quelques problèmes à propos d'un site Web.- Vous venez de concevoir un site WEB. Vous le mettez en service. Vous attendez quelque temps. Vous vous posez tout naturellement les questions suivantes : mon site est-il souvent consulté ? quelles sont les pages les plus consultées ? comment les utilisateurs accèdent-ils à ces pages ? le cheminement vers ces pages est-il celui auquel j'avais pensé ? faut-il revoir la hiérarchie des pages ?

Où est la réponse ?.- Les logiciels de maintien de site WEB, en particulier *Apache*, le plus utilisé, permettent de répondre immédiatement à un certain nombre de ces questions, mais pas à toutes.

La dernière question est une question des plus subjectives et il n'est certainement pas du propos d'un tel outil d'y répondre.

L'avant-dernière question, par contre, peut se transformer en une question objective :

Quel est le cheminement suivi par les utilisateurs pour atteindre telle page ?

à laquelle on aimerait bien une réponse immédiate. En fait les logiciels de maitien des sites Web ne peuvent pas répondre à cette question, et ceci pour une raison intrinsèque.

Est-ce un coup du CNIL ?.- Remarquons d'abord que, pour une raison d'ordre éthique, même si on pouvait techniquement y répondre, nous aurions toutes les chances que cela ne soit pas permis par la CNIL (*Commission Nationale de l'Informatique et des Libertés*) en France ou un organisme analogue dans un autre pays. Mais en fait ce n'est pas la raison essentielle, il s'agit bien d'un vrai problème technique.

Limitations techniques.- Voyons d'abord les renseignements les plus proches de ceux que l'on voudrait et qui sont immédiatement disponibles avec les logiciels de maintien des sites Web. En lui demandant de faire l'*historique des consultations*, on obtient un listing d'un grand nombre de lignes de la forme suivante :

```
194.214.21.124 - - [19/Jan/2000:13:44:58 +0100] ''GET /~cohen/ HTTP/1.0'' 200 339
```

Nous n'allons pas détailler le format de ces lignes. On comprend tout de suite que l'on a le *numéro IP* de la machine appelante, la date et l'heure ainsi qu'une référence à la page consultée. La date et l'heure sont données sous un format presque compréhensible (on remarquera l'indication de la correction par rapport à l'heure GMT). Qu'importe le format de désignation des pages, il suffit de savoir que l'on peut y faire référence. Reste le numéro IP.

Qu'est-ce qu'un numéro IP ?.- Je sais bien qu'il s'agit presque d'un affront que de penser que vous ne savez pas de quoi il retourne. Mais nous avons besoin d'éclaircir ce point pour bien comprendre le problème auquel nous sommes confrontés. Chaque machine reliée à Internet reçoit un numéro unique, dit *num éro IP* (pour *Internet Protocol*), qui est un uméro d'identification comme notre numéro d'identité nationale (dit de sécurité sociale), en quatre parties séparées par un point.

Le problème du numéro IP.- Au début de l'utilisation des réseaux, c'était simple : un numéro IP correspondait à un ordinateur donné, plus précisément à un mini-ordinateur. Sur une période suffisamment courte, un numéro IP correspondait donc à un utilisateur donné. Mais le dévelopement de l'utilisation des réseaux a changé cette façon d'attribuer les numéros IP.

La plupart des utilisateurs d'Internet passent par une FAI (*Fournisseur d'Acc ès Internet* ; en anglais IAP pour *Internet Access Provider*). On se connecte à ces FAI le plus souvent par le truchement d' une ligne téléphonique grâce à un modem. L'ordinateur de l'utilisateur dispose alors d'un numéro IP distribué de façon dynamique (c'est-à-dire qu'il ne sera pas le même d'une session à l'autre) mais de l'extérieur ce n'est pas ce numéro qui sera visible, mais celui du garde-barrière du FAI. Si bien que le numéro IP obtenu dans la ligne ci-dessus est celui du FAI et non celui de l'utilisateur.

Plusieurs utilisateurs du même FAI auront donc le même numéro IP. Vous voyez au passage que cela simplifie le travail de la CNIL. Si quelqu'un détecte la consultation d'une page Web contraire aux bonnes mœurs ou non politiquement correcte, l'utilisateur à qui on le reprocherait peut toujours se disculper en prétendant que ce n'était pas lui.

Un modèle.- Notre premier but (qui n'est pas notre but premier) est de concevoir un modèle permettant d'obtenir une réponse probable à la question que nous avons prise en exemple.

Je voudrais savoir, par exemple, si ayant atteint la page unetelle, l'utilisateur va, parmi celles qui lui sont proposées, à celle-ci, de celle-ci à celle-là et ainsi de suite.

Peut-on savoir, toujours avec les outils de maintien des sites Web, lorsqu'un utilisateur a demandé telle page quelle page il demande ensuite ? La réponse est non, malheureusement : les références sont rapatriées chez lui, il a toute latitude de demander une page de notre site ou non, le seul retour que nous en aurons est une ligne du format indiqué plus haut. S'il s'agit d'un FAI ayant très peu de requêtes sur notre site, on peut faire l'hypothèse qu'il n'y a qu'un seul utilisateur qui s'intéresse à nous et le suivre à la trace.

Il en est en fait rarement ainsi. De nombreux utilisateurs du même FAI (en tous les cas de même numéro IP) interrogent un site donné presque en même temps et leurs requêtes s'entrecroisent.

Nous pouvons cependant concevoir un modèle assez réaliste qui va nous permettre de répondre à la question. Disons qu'il faut une certaine unité de temps pour faire une requête et recevoir la réponse. Voilà mon modèle :

Je dirai qu'un utilisateur a consulté les pages 1, 2, 3, 4 dans cet ordre si, lors de cinq ou six unités de temps, je retrouve sur mon historique le même numéro IP avec la consultation des pages 1, 2, 3, 4 dans cet ordre.

Je peux avoir eu la suite de consultation 1, 2, 2, 1, 3, 2, 4, par exemple, dans ma *fenêtre* (c'est ainsi que l'on appellera le nombre d'unités de temps pris en référence). On retrouve bien la sous-suite 1, 2, 3, 4. On y retrouve aussi d'autres sous-suites telles que 2, 1, 3, 2.

2 Le problème formel

Le modèle étant trouvé, formalisons-le. Commençons par quelques définitions.

DÉFINITIONS 1.- *Un* alphabet *est un ensemble fini non vide A.*

Un mot *de* longueur n *sur l'alphabet A est une application t de l'intervalle d'entiers naturels $\{1, \ldots, n\}$ dans A.*

Le seul mot de longueur nulle est le mot vide, *noté ε. Un mot non vide $t : i \mapsto t_i$ sera noté $t_1 t_2 \cdots t_n$.*

Un langage *sur l'alphabet A est un ensemble de mots sur l'alphabet A.*

Soit $t = t_1 t_2 \cdots t_n$ un mot. Un mot $p = p_1 p_2 \cdots p_k$ est un sous-mot *(ou* facteur*) de t si, et seulement si, il existe un entier j tel que $t_{j+i} = p_i$ pour $1 \leq i \leq k$.*

Une fenêtre *de* taille *w sur le mot t, en abrégé une w-*fenêtre*, est un sous-mot $t_{i+1} t_{i+2} \cdots t_{i+w}$ de t de longueur w ; il y a $n - w + 1$ telles fenêtres.*

Un mot p est une sous-suite *de t si, et seulement si, il existe des entiers $1 \leq i_1 < i_2 < \cdots < i_k \leq n$ tels que $t_{i_j} = p_j$ pour $1 \leq j \leq k$. Si p est une sous-suite de t et si nous avons $i_k - i_1 < w$, alors p est une* sous-suite *de t* dans une w-fenêtre.

Exemple.- Si t = "researcher" alors "sea" est un sous-mot de t, donc "sea" est aussi une sous-suite de t. Par contre, "see" n'est ni un sous-mot, ni une sous-suite de t dans une 6-fenêtre, mais "see" est une sous-suite de t dans une 7-fenêtre. Voir la figure 1.

r	*e*	s	e	*a*	*r*	*c*	*h*	e	*r*

FIG. *1* *. Une 7-fenêtre avec un sous-mot minimal contenant "see".*

Donnons des noms techniques à quelques problèmes liés aux mots et essayons de reconnaître le nôtre, c'est-à-dire celui qui correspond à notre modèle.

DÉFINITIONS 2.- *Étant donnés un alphabet A et des mots p et t sur A :*

- *le* problème de l'appariement *(en anglais* pattern-matching problem*) consiste à trouver si p est un sous-mot de t,*
- *le* problème de la sous-suite *(en anglais* plain subsequence matching problem*) consiste à trouver si p est une sous-suite de t,*
- *étant donné de plus un entier w, dit* taille de la fenêtre,
  - *le* problème de l'existence d'une sous-suite dans une fenêtre *(*WESP *en anglais pour* Window-Existence Subsequence matching Problem*) consiste à trouver si p est une sous-suite de t dans une w-fenêtre.*
  - *le* problème du comptage de sous-suites dans une fenêtre *(*WASP *en anglais pour* Window-Accumulated Subsequence matching Problem*) consiste à dénombrer* toutes *les w-fenêtres dans lesquelles p est une sous-suite de t.*

On voit que le WASP est une modélisation de notre problème initial. Le problème de l'appariement des mots est évidemment utilisé par les traitements de texte et les éditeurs de texte lorsqu'on utilise l'outil `recherche`, d'où sa grande importance.

3 Un petit détour vers les traitements de texte

Un autre problème.- Nous venons d'énoncer formellement notre problème. Nous avons vu qu'il est proche d'un problème qui se pose pour les éditeurs de texte, les traitements

de texte ou les outils du système d'exploitation lui-même (recherche d'un fichier), à savoir celui de la *recherche* d'occurrences de sous-mots.

Une solution naïve.- Si on demande à un étudiant un programme permettant de résoudre ce problème, il y a beaucoup de chances qu'il propose (s'il propose quelque chose) ce que l'on appelle la *solution naïve*, l'adjectif qualificatif est suffisamment explicite.

Voilà en quoi consiste cet algorithme. On considère le texte comme un (grand) tableau de caractères, chaque lettre étant repérée par un numéro. Un index permet de se positionner sur chacune des lettres de 0 à $n-1$, si n est la longueur du texte. Si la lettre ainsi pointée est égale à la première lettre du motif, on sauvegarde l'index ; on regarde alors si la lettre suivante est la seconde lettre du motif, s'il n'en est pas ainsi on revient à la valeur sauvegardée de l'index ; sinon on regarde si la lettre suivante est égale à la seconde lettre du motif et ainsi de suite ; si les k lettres sont égales alors on a trouvé une occurrence du motif.

Complexité de cet algorithme.- On est toujours content lorsqu'on a trouvé un algorithme. Reste à savoir s'il est très efficace. Il n'est pas difficile de s'apercevoir que, dans le pire cas, ici une occurrence du motif à la fin ou pas d'occurrence du tout, on a nk (ou mieux $O(nk)$) déplacements d'index à effectuer, si k est la longueur du motif p et n la longueur du texte t.

Un exemple.- Pour savoir si l'algorithme est efficace, considérons un exemple. Prenons la dernière version de l'*Encyclopædia universalis* (trois CD-ROM ou un DVD-ROM) et cherchons le nombre d'occurrences du mathématicien Yuri Matiiassevitch.

Disons que le texte occupe un CD-ROM complet, soit 540 Mo, donc $n = 54.10^7$. Le motif prend 19 lettres, pour simplifier disons $k = 20$. Disons que mon super micro-ordinateur est capable de faire dix millions de déplacements d'index à la seconde (ce qui est très optimiste). Il faut donc $54.10^7.20/10^7 = 1080s$, soit 18 minutes pour effectuer ma recherche.

Un meilleur algorithme.- Knuth, Morris et Pratt [KMP77] ont donné dans un célèbre article écrit en 1977 un algorithme pour résoudre le problème de l'appariement de motif en temps linéaire $O(n)$. Qu'importe en quoi il consiste pour notre propos.

Reprenons notre exemple. En imaginant que la constante de notre O soit la même, nous résolvons le problème en $54.10^7/10^7 = 54s$.

On voit sur cet exemple l'intérêt de la recherche de meilleurs algorithmes.

4 Résolution de notre problème

Qu'en est-il des algorithmes pour notre problème, le WASP?

4.1 Algorithme naïf

Je vous laisse expliciter, à titre d'exercice, une solution naïve pour le *WASP*. Elle est en temps $O(nkw)$, où w est la taille de l'alphabet. Je vous laisse imaginer l'efficacité dans les mêmes conditions que l'exemple précédent avec l'alphabet ASCII étendu de 256 lettres.

4.2 Algorithme standard

Il existe un algorithme plus élaboré, que nous appelons l'**algorithme standard**, qui est en $O(nk)$ (voir [MTV95,DFGGK97]). Explicitons-le.

Idée principale.- Soient $p = p_1 p_2 \dots p_k$ le motif et $t = t_1 t_2 \dots t_n$ le texte. Remarquons que le problème de recherche de sous-suite se réduit à trouver le nombre de w-fenêtres de t qui contiennent un *sous-mot minimal* contenant p. Bien entendu un sous-mot de t contenant p est dit *minimal* s'il ne possède pas lui-même de sous-mot strict contenant p.

Exemple.- Soient $t =$ `researshers` (*sic*) et $p =$ `se`. Dans ce cas `rese` et `rshe` sont des sous-mots contenant `se`, mais ce *ne sont pas* des sous-mots minimaux contenant `se`. Par contre `se` et `she` sont des sous-mots minimaux contenant `se`.

DÉFINITION.- *Un mot $t_r \dots t_s$ est un* sous-mot minimal *de t* contenant $p_1 \dots p_l$ *si, et seulement si :*

1. il existe des entiers $r \leq i_1 < i_2 < \dots < i_l \leq s$ tels que $t_{i_j} = p_j$ pour $1 \leq j \leq l$ (on dit que $i_1, i_2, \dots, i_l$ est une occurrence *de $p_1 \dots p_l$ dans $t_r \dots t_s$),*

2. et, de plus, aucun sous-mot strict de $t_r \dots t_s$ ne contient $p_1 \dots p_l$, c'est-à-dire que pour toutes les occurrences de $p_1 \dots p_l$ dans $t_r \dots t_s$, on a $i_1 = r$ et $i_l = s$.

En fait toute fenêtre contenant p comme sous-suite contient aussi un sous-mot minimal contenant p (voir la figure 1).

Idée de l'algorithme.- Afin de compter le nombre de w-fenêtres de t contenant un sous-mot minimal contenant p, nous nous servons d'un tableau d'entiers $s[1...k]$, où $s[l]$ est la position de départ de t du sous-mot minimal contenant $p_1 \dots p_l$ qui a été vu le plus récemment, s'il existe et $-\infty$ sinon.

Nous initialisons $s[1...k]$ à $[-\infty \dots -\infty]$.

L'algorithme se déroule en déplaçant une w-fenêtre sur t. Soit i l'index de t correspondant à la fin de la fenêtre en cours. Si $s[l] = j \neq 0$, ceci signifie qu'il existe un sous-mot minimal contenant $p_1 \dots p_l$ débutant à l'index j dans le sous-mot $t_j \dots t_i$. Ainsi si on a $s[k] = j$ et $i - j < w$, on en conclut que la fenêtre en cours contient un sous-mot minimal contenant p.

La variable *count* enregistre le nombre de w-fenêtres contenant p comme sous-suite.

Exemple.- Soient $t =$ `researshers`, $p =$ `see` et $w = 8$. Nous obtenons la table suivante :

i	t	$s[1]$	$s[2]$	$s[3]$	$i - s[3]$	accepte?
1	r	$-\infty$	$-\infty$	$-\infty$	∞	non
2	e	$-\infty$	$-\infty$	$-\infty$	∞	non
3	s	3	$-\infty$	$-\infty$	∞	non
4	e	3	3	$-\infty$	∞	non
5	a	3	3	$-\infty$	∞	non
6	r	3	3	$-\infty$	∞	non
7	s	7	3	$-\infty$	∞	non
8	h	7	3	$-\infty$	∞	non
9	e	7	7	3	6	oui
10	r	7	7	3	7	oui
11	s	11	7	3	8	non

l'acceptation se faisant lorsque $i - s[3] < 8$.

Algorithme.- Donnons l'algorithme dans un pseudo-langage. Dans celui-ci "`DO forall` $l \in I$ `inst ENDDO`" signifie qu'il faut effectuer `inst` simultanément pour tous les indices $l \in I$.

```
DO forall l ∈ [1...k] s[l] := 0 ENDDO
count := 0
DO for i ∈ [1...n] in increasing order
    IF t_i = p_1 then s[1] := i ENDIF
    DO forall l ∈ [2...k]
        IF t_i = p_l then s[l] := s[l − 1] ENDIF ENDDO
    IF i − s[k] < w then count := count + 1 ENDIF
ENDDO
```

Complexité.- Il est facile de voir que la complexité est améliorée, puisque nous trouvons maintenant $O(nk)$.

Dans [M97], Mannila poe la question de savoir s'il existe un algorithme pour *WASP* en $o(nk)$.

4.3 Le meilleur algorithme

En fait il existe un algorithme linéaire. Il a été trouvé par Yuri Matiiassévitch, un collègue de l'université Paris VII, une collègue de l'université Paris VI et l'auteur [BCGM99].

Je renvoie à l'article original pour les détails de celui-ci, ce qui demande peu de prérequis.

Références

[BCGM99] L. Boasson, P. Cegielski, Irène Guessarian, Yuri Matiyasevich, *Window-Accumulated Subsequence Matching Problem is Linear*, PODS 1999, ACM Press, pp. 327–336.

[DFGGK97] G. Das, R. Fleischer, L. Gąsienic, D. Gunopoulos, J. Kärkkäinen, *Episode Matching*, Proc. 1997 Combinatorial Pattern Matching Conf., LNCS 1264, Springer-Verlag, Berlin (1997), pp. 12–27.

[KMP77] D. Knuth, J. Morris, V. Pratt, *Fast Pattern Matching in Strings*, SIAM Journal of Comput. Vol 6(2), (1977), pp. 323–350.

[M97] H. Mannila, *Methods and Problems in Data Mining*, Proc. 1997 ICDT Conf., LNCS 1186, Springer-Verlag, Berlin (1997), pp. 41–55.

[MTV95] H. Mannila, H. Toivonen, A. Verkamo, *Discovering Frequent Episodes in Sequences*, Proc. 1995 KDD Conf., (1995), pp. 210–215.