Algorithmique I
Année 2008-2009

A. Benoit, B. Depardon
C. Mouilleron, C. Rezvoy

DM - Des histoires de tableaux

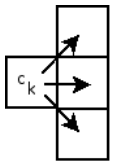
À rendre pour le 22 octobre 2008 (tout retard sera sanctionné.)

Prenez le temps de rédiger les preuves et ne tentez pas de bluff ! En cas de question ou de problème, envoyer un mail à christophe.mouilleron@ens-lyon.fr ou passer à mon bureau. Notamment, la partie 3 de l'exercice 3 demande d'avoir des notions de base sur les automates. Prévenez moi si vous n'avez pas ces notions et que les rappels en annexe ne vous suffisent pas !

Exercice 1. *Pour s'échauffer...*

On se donne un tableau d'entiers T à n lignes et m colonnes. On appelle chemin à travers le tableau T une succession de cases $(c_1, \dots, c_m)$ telle que :

- la case c_k est dans la colonne k du tableau T, pour tout k
- pour tout k, les cases c_k et c_{k+1} sont adjacentes (c_{k+1} est soit 1 cran à droite et 1 cran au dessus de c_k, soit juste à droite de c_k, soit 1 cran à droite et 1 cran en dessous de c_k)



On notera $T[c_k]$ l'entier de T correspondant à la case c_k.

Le poids d'un chemin traversant T est alors défini par $w(\mathbf{c}) = \sum_{k=1}^{m} T[c_k]$.

1 - Proposer un algorithme qui calcule un chemin traversant T de poids minimal.

2 - Illustrer le déroulement de l'algorithme sur le tableau suivant :

$$T = \begin{array}{|c|c|c|c|c|c|}\hline 3 & 4 & 1 & 2 & 6 & 8 \\ \hline 6 & 1 & 8 & 2 & 7 & 9 \\ \hline 5 & 9 & 3 & 9 & 1 & 5 \\ \hline 8 & 4 & 1 & 3 & 2 & 6 \\ \hline 3 & 7 & 2 & 8 & 6 & 4 \\ \hline \end{array}$$

Exercice 2. *Diviser pour régner ?*

On se donne deux ensembles de nombres $\{x_1, \dots, x_n\}$ et $\{y_1, \dots, y_m\}$ tels que :
- $x_1 < x_2 < \cdots < x_{n-1} < x_n$
- $y_1 < y_2 < \cdots < y_{m-1} < y_m$.

On considère le tableau T de taille $n \times m$ défini par $T[i,j] = x_i + y_j$. Le problème consiste à chercher si un nombre donné N appartient au tableau T.

1 - Écrire un algorithme (naïf) pour résoudre le problème. Quel est sa complexité en nombre de comparaisons ?

2 - Proposer un algorithme utilisant le paradigme "diviser pour régner". Prouver sa correction et calculer sa complexité dans le cas où $n = m$.

3 - En réalité, il est possible de faire mieux en utilisant astucieusement la croissance des x_i et des y_j. Proposer un algorithme en $O(n+m)$ pour résoudre le problème.

Exercice 3. *Recherche de motif dans un tableau.*

Le but de cet exercice est d'arriver à un algorithme efficace pour rechercher les occurrences d'un motif M dans un tableau T à deux dimensions. Un exemple est présenté à la figure 1.

$$\Sigma = \{a, b\} \qquad M = \begin{array}{|c|c|c|}\hline a & a & a \\ \hline b & b & a \\ \hline a & a & b \\ \hline \end{array} \qquad T = \begin{array}{|c|c|c|c|c|c|c|}\hline a & b & a & b & a & b & b \\ \hline a & a & a & a & b & b & b \\ \hline b & b & b & a & a & a & b \\ \hline a & a & a & b & b & a & a \\ \hline b & b & a & a & a & b & b \\ \hline a & a & b & a & a & a & a \\ \hline \end{array}$$

FIG. 1 – Problème : trouver toutes les occurrences du motif M dans le tableau T

En fait, on s'intéressera d'abord au cas de la recherche d'un motif dans une chaîne de caractères (= un tableau à une dimension), puis à la recherche d'un motif dans un tableau (à 2 dimensions). L'ensemble des caractères sera noté Σ. On suppose que Σ a d éléments avec $d \geq 2$.

En dimension 1, M sera de taille m et T de taille n, et il s'agira de trouver tous les indices s tel que $M[1..m] = T[s+1..s+m]$ (*i.e.,* tel que $M[i] = T[s+i]$ pour $1 \leq i \leq m$).

En dimension 2, M sera de taille $m \times m'$ et T de taille $n \times n'$, et il s'agira de trouver tous les couples (r, s) tels que $M[1..m, 1..m'] = T[r+1..r+m, s+1..s+m']$.

1 - Performances des algorithmes naïfs.

1.1 - Proposer un algorithme naïf pour rechercher un motif dans une chaîne de caractères. Quel est **précisément** le nombre de comparaisons dans le pire cas ? Donner un pire cas.

1.2 - (plus dur) Montrer que si la chaîne de caractères et le motif sont choisis aléatoirement, alors le nombre moyen de comparaisons dans l'algorithme naïf est $(n-m+1)\dfrac{1-d^{-m}}{1-d^{-1}} \leq 2(n-m+1)$.

1.3 - Généraliser l'algorithme précédent pour la recherche d'un motif dans un tableau. Étudier le nombre de comparaisons dans le pire cas.

Désormais, on ne s'intéressera plus qu'à la complexité dans le **pire** cas.

2 - Recherche d'un motif dans une chaîne de caractères.

L'inconvénient principal dans l'algorithme naïf vient du fait qu'on ne tire pas pleinement profit des comparaisons que l'on effectue. En fait, plutôt que d'avancer de 1 en 1 dans la chaîne de caractères T, on voudrait faire des sauts plus grands lorsque c'est possible. Pour cela, on va introduire la fonction w définie par :

- $w[0] = -1$
- $w[j] =$ le plus grand $k \in \{1, \ldots, j-1\}$ tel que $M[1..k]$ soit un suffixe propre de $M[1..j]$ (*i.e.,* tel que $M[1..k] = M[j-k+1..j]$)
  Si aucun tel k n'existe, on prendra $w[j] = 0$.

2.1 - On propose de calculer la fonction w à l'aide de l'algorithme 1. Justifiez la correction de l'algorithme. Quel est son coût en nombre de comparaisons ?

Indication : Formuler des invariants précis pour chacune des boucles et raisonner par récurrence.

Algorithme 1 : calcul-w

Entrée : Motif M de taille m
Sortie : Fonction w

début
1 $w[0] \leftarrow -1$
2 $t \leftarrow -1$
3 **pour** j *de* 1 *à* m **faire**
4 **tant que** $t \geq 0$ *et* $M[t+1] \neq M[j]$ **faire** $t \leftarrow w[t]$
5 $t \leftarrow t+1$
6 $w[j] \leftarrow t$
7 Retourner w
fin

2.2 - À l'aide de cette fonction w, écrire une variante de l'algorithme naïf pour la recherche des occurrences du motif M dans la chaîne de caractères T. Quel est le coût d'une recherche ?

2.3 - (dur) En fait, la fonction w ne tient toujours pas en compte de toute l'information apportée par les comparaisons. Proposer une meilleure fonction w'. Donner un algorithme pour calculer w' et illustrer le gain par rapport à w sur un exemple.

Indication : Que pensez vous du cas où $M[k+1] = M[j+1]$ dans la définition de $w[j]$?

3 - Automates et recherches multiples.

En réalité, la recherche du motif $M[1..m]$ dans le tableau $T[1..n]$ est liée à la reconnaissance du langage $L = \Sigma^* M[1..m]$.

3.1 - Proposer un algorithme qui construit un automate déterministe complet reconnaissant le langage L à partir de la fonction w. Donner sa complexité.

3.2 - Proposer alors un algorithme permettant de rechercher toutes les occurrences du motif M dans T à l'aide d'un automate. Quel désavantage y a t'il à passer par un automate plutôt que par la fonction w directement ?

3.3 - (dur) On peut généraliser cette approche pour rechercher **simultanément** plusieurs motifs. Par exemple, si $\Sigma = \{a, b\}$ et qu'on désire rechercher les motifs *aba* et *aab*, on préférera utiliser l'automate de la figure 2 plutôt que d'utiliser deux automates et de faire deux parcours de la chaîne de caractères successifs.

Proposer une généralisation w'' de la fonction w pour traiter le cas de plusieurs motifs $M_1, M_2, \ldots, M_k$, ainsi qu'un algorithme pour calculer cette nouvelle fonction et construire l'automate permettant une recherche simultanée de ces motifs. Montrer que le nombre de comparaisons et le coût en mémoire pour construire l'automate sont en $O(\sum_{j=1}^{k} m_j)$.

Indications :
- *Dans quel ordre doit-on calculer les $w''(j)$?*
- *Pourquoi est-ce plus simple si on suppose qu'aucun M_i n'est un sous motif d'un autre M_j ?*

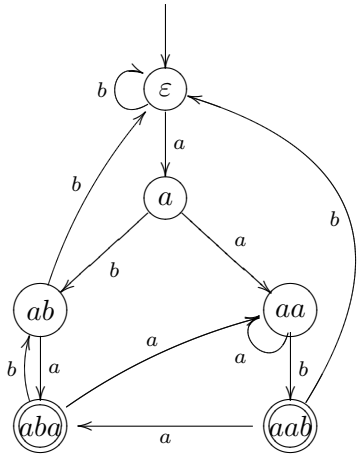


FIG. 2 – Automate permettant la recherche simultanée des motifs aba et aab

4 - Recherche efficace d'un motif dans un tableau.

On va d'abord raisonner sur l'exemple de la figure 1.

L'idée de départ est de considérer chaque colonne du motif M comme un mot et de chercher simultanément ces mots dans les colonnes du tableau T.

4.1 - Grâce à l'automate de la figure 2, on peut associer à chaque case $T[i, j]$ un état de l'automate en parcourant T colonne par colonne de haut en bas. Donner cette correspondance.

4.2 - Que reste t'il à faire pour trouver les occurrences de M dans T ? Illustrer sur l'exemple.

4.3 - Donner l'algorithme complet de recherche d'un motif M dans un tableau T. Quelle est sa complexité en nombre de comparaisons ? De combien de mémoire supplémentaire a-t-on besoin ?

Annexe : Rappel sur les automates.

Schématiquement, un automate déterministe complet est un ensemble d'états représentés par des ronds ((a) par exemple). Parmi ces états, il y a un état initial pointé par une flèche ($\longrightarrow(\varepsilon)$) et des états finaux représentés avec des ronds doubles (comme $((aba))$). Ces états sont reliés par des flèches étiquetées par un caractère de Σ ($\xrightarrow{a}$). De plus, on impose que de chaque état parte exactement une flèche par caractère de Σ.

Si on se donne un mot, on peut alors partir de l'état initial, lire les lettres une par une et suivre à chaque lettre la flèche correspondante. Si à la fin du mot, l'état courant est final, on dit que le mot est reconnu par l'automate. Par exemple, le mot *abbaba* est reconnu par l'automate de la figure 2 car on termine à l'état final *aba*.

L'ensemble des mots reconnu par l'automate est appelé langage reconnu par l'automate. Dans la partie 3, le langage que nous voulons reconnaître est $L = \Sigma^* M[1..m]$. Ce langage contient tous les mots finissant par $M[1]M[2]\dots M[m]$ (Σ^* désigne en fait un nombre quelconque de caractères de Σ).