Dea Algorithmique

Paradigmes de la combinatoire et de l'algorithmique

Corrigé du premier problème d'entraînement

Septembre 2002

Les sous-mots

I. Généralités

Question 1. *Donner un algorithme qui teste si le mot u de longeur k est sous-mot du mot f de longueur n.*

```
static boolean estSousMot (int[] g, int[] f){
    int i = 1; int j = 1; int n = f.length - 1;
    int m = g.length - 1;
    while ( j <= m && i <= n) {
        if (g[j] == f[i]) j++;
        i++;}
    return (j > m);
}
```

Question 2. *Combien un mot de longueur k peut-il avoir de sous-mots différents au maximum ? Et au minimum ?*

Si le mot f a toutes ses lettres différentes il a 2^k sous-mots distincts, s'il est égal à a^k il n'a que $k+1$ sous-mots différents.

Question 3. *Quel est le nombre maximum de sous-mots incomparables de abab ? Quel est le nombre maximum de sous-mots incomparables d'un mot f de longueur n ?*

Les sous-mots de *abab* sont

$$\{a, b, ab, aa, bb, ab, ba, aab, aba, abb, bab, abab\}$$

Il y a 4 sous-mots incomparables au maximum : ceux de longueur 2 ou ceux de longueur 3. Dans un mot de longueur n dont toutes les lettres sont différentes tous les sous-mots de longueur $\lfloor \frac{n}{2} \rfloor$ sont incomparables, il y en a $\binom{n}{\lfloor \frac{n}{2} \rfloor}$; il est raisonable de penser, bien que la démonstration ne soit pas immédiate que ce nombre est le maximum possible.

Question 4. *Montrer que si A est un alphabet de taille k, alors la connaissance de $\pi_{a,b}(f)$ pour toutes les $\frac{k(k-1)}{2}$ paires de lettres $\{a, b\}$ de A permet de déterminer f de manière unique.*

La première lettre du mot f est le x tel que tous les $\pi_{x,y}(f)$ commencent par x ; une fois cette première lettre déterminée on itère en la supprimant de tous les $\pi_{x,y}(f)$.

On ne peut se passer de l'un des $\pi_{a,b}(f)$ pour effectuer cette détermination, car on ne pourra pas distinguer les mots qui commencent par a de ceux qui commencent par b sans connaître $\pi_{a,b}(f)$.

II. Coefficients binomiaux

Question 1. Valeurs de

$$\binom{a^pb^qa^r}{ab} = pq, \qquad \binom{a^pb^qa^r}{aba} = pqr, \qquad \binom{(ab)^p}{ab} = \frac{p(p+1)}{2}$$

Question 2. *Démontrer la formule suivante, dans laquelle f, g sont deux mots et a, b deux lettres ($\delta_{a,b}$ vaut 1 si $a = b$ et 0 sinon) :*

$$\binom{fa}{gb} \;=\; \binom{f}{gb} \;+\; \delta_{a,b}\binom{f}{g}$$

Si a est différent de b les positions où gb est sous-mot de fa sont toutes contenues dans f donc

$$\binom{fa}{gb} \;=\; \binom{f}{gb}$$

Si $a = b$ les positions où gb est sous-mot de fa sont soit toutes contenues dans f, soit correspondent au choix de la dernière lettre a dans fa pour être couplée avec le b de gb on a ainsi

$$\binom{fa}{gb} \;=\; \binom{f}{gb} \;+\; \binom{f}{g}$$

Question 3. La construction du triangle de Pascal est le cas particulier de cette formule pour $f = a^n$ et $g = a^p$, on a alors :

$$\binom{n}{p} \;=\; \binom{n-1}{p-1} \;+\; \binom{n-1}{p}$$

Question 4. *Donner un algorithme récursif de calcul de $\binom{f}{g}$ qui utilise cette formule.*

```
static int binome (int[] g , int f[], int m, int n){
    compte++;
    if (n == 0) return 0;
    if (m == 0) return 1;
    if (f[n] == g[m]) return (binome (g, f ,m-1,n-1)
                              + binome (g, f, m, n-1));
    return binome(g, f, m, n-1);
}
```

Le nombre d'appels de fonctions qui est effectué par cet algorithme en fonction des longueurs n et m respectivement de f et de g est $\binom{n}{m}$.

Un algorithme itératif qui effectue le calcul en utilisant un tableau de n lignes et m colonnes est simple :

```
static int binome (int[] g , int f[], int m, int n){
    int[][] binomial = new int[n+1][m+1];
    for (int j = 0; j <= n; ++j) binomial[0][j] = 0;
    for (int i = 1; i <= n; ++i){
       binomial[i][0] = 1;
```

```
        for (int j = 0; j <= n; ++j){
           if (f[i] == g[j])
              binomial[i][j] = binomial[i-1][j-1] + binomial[i-1][j];
           else     binomial[i][j] =  binomial[i-1][j];
}
```

Question 5. La formule :

$$\binom{fg}{h} = \sum_{uv=h} \binom{f}{u}\binom{g}{v}$$

Découle de ce que chaque fois que l'on a une occurence de h dans fg cela correspond à une décomposition de h en $h = uv$ où u est un sous-mot de f et v un sous-mot de g.

$$\sum_{h \in A^*} (-1)^{|h|} \binom{f}{h}\binom{h}{g} = (-1)^{|f|}\delta_{f,g}$$

On vérifie que si g n'est pas sous mot de f alors on obtient 0 à gauche et à droite du signe =.

Si $f = g$ on trouve 1 des deux cotés.

Si g est sous-mot de f, cette formule se démontre plus simplement si f a toutes ses lettres distinctes, ce que l'on peut supposer en changeant d'alphabet. $\sum_{h \in A^*} (-1)^{|h|} \binom{f}{h}\binom{h}{g}$ dans ce cas correspond aux sous-ensembles de $\{1, \ldots n\}$ qui contiennent les occurrences de g ; le nombre de sous-ensembles de taille $i + m$ est $\binom{n-m}{i}$ On obtient donc :

$$\sum_{i=0}^{n-m} (-1)^{i+m} \binom{n-m}{i} = (-1)^m \sum_{i=0}^{n-m} (-1)^i \binom{n-m}{i}$$

Or ceci correspond à la valeur prise par $(1 + x)^{n-m}$ pour $x = -1$ ce est égal à 0 sauf si $n = m$.

III. Plus longs sous-mots communs

Question 1.

$\lambda(fa, gb)$ s'exprime à l'aide de $\lambda(f, gb), \lambda(fa, g)$ et $\lambda(f, g)$. On a ainsi :

Si $a = b$ $\lambda(fa, gb) = 1 + \lambda(f, g)$.

Sinon $\lambda(fa, gb) = max(\lambda(fa, g), \lambda(f, gb))$.

Question 2. *En déduire un algorithme qui calcule le tableau* `lambda[i][j]` *des longueurs des plus longs sous-mots communs à f_i et g_j,*

```
static int[][] consTab(int[] f, int[] g){
    int n = f.length -1;
    int m = g.length -1;
    int[][] lambda = new int[n+1][m+1];
    for (int i = 0; i <= n ; ++i) lambda[i][0] = 0;
    for (int j = 0; j <= m ; ++j) lambda[0][j] = 0;
    for (int i = 1; i <= n ; ++i)
        for (int j = 1; j <= m ; ++j)
            if (f[i] == g[j]) lambda[i][j] = lambda[i-1][j-1]+1;
            else if (lambda[i-1][j] > lambda[i][j-1])
                         lambda[i][j] = lambda[i-1][j];
            else  lambda[i][j] = lambda[i][j-1];
    return lambda;
}
```

Question 3. *Construire le tableau pour les mots $f = abbababba$ et $g = aaabbab$. Utiliser le tableau ainsi calculé pour obtenir un plus long sous-mot commun à ces deux mots.*

Question 4. *Donner un algorithme qui calcule un plus long sous-mot commun en utilisant le tableau* `lambda[i][j]`.

```
static int[] maxSousMot (int[] f, int[] g){
    int[][] t = consTab(f,g);
    int n = f.length -1;
    int m = g.length -1;
    int k = t[n][m];
    int[] h = new int[k+1];
    int i = n; int j = m;
    while (i > 0 && j >0) {
        if (f[i] == g[j]) {
            h[k] = f[i]; i--; j--; k--;}
        else if (t[i][j] == t[i-1][j]) i--;
        else j--;
    }
    return h;
}
```

Question 5. *Proposer un nouvel algorithme de calcul de la longueur du plus long sous-mot commun ayant la même complexité en temps que le précédent et qui n'utilise que deux tableaux d'une seule ligne de longueur $min(n, m)$ au lieu du tableau* `lambda[i][j]` *à deux entrées.*

```
static int [] maxLong (int[] f, int n ,  int[] g ,int m){

    int[][] lambda = new int[2][m+1];
    for (int i = 0; i <= 1 ; ++i) lambda[i][0] = 0;
    for (int j = 0; j <= m ; ++j) lambda[0][j] = 0;
    for (int i = 1; i <= n ; ++i){
        for (int j = 1; j <= m ; ++j)
            if (f[i] == g[j]) lambda[1][j] = lambda[0][j-1]+1;
            else if (lambda[0][j] > lambda[1][j-1])
                        lambda[1][j] = lambda[0][j];
            else  lambda[1][j] = lambda[1][j-1];
        for (int j = 1; j <= m ; ++j) lambda[0][j] = lambda[1][j];
    }
     return lambda[1];
}
```

On se propose de donner un algorithme de calcul du plus long sous-mot commun qui fonctionne en temps $O(mn)$ et en espace $O(min(n, m))$. Pour cela on utilise la méthode diviser pour régner et on se propose de commencer par calculer le plus long sous-mot commun à f_p et g_k où p est égal à la partie entière de $n/2$ et k un entier à déterminer. Celui-ci doit être tel que

$$\lambda(f, g) = \lambda(f_p, g_k) + \lambda(f'_p, g'_k) \text{ avec } f = f_p f'_p, \quad g = g_k g'_k$$

Question 6. Pour déterminer l'entier k, on calcule pour chaque couple i, j tels que $p \leq i \leq n$ et $1 \leq j \leq m$ l'entier $\ell = \mu(i, j)$ satisfaisant :

$$\lambda(f_i, g_j) = \lambda(f_p, g_\ell) + \lambda(f''_i, g''_j)$$

avec $f_i = f_p f''_i, g_j = g_{ell} g''_j$.

On peut exprimer $\mu(i,j)$ en fonction de $\mu(i-1,j),\ \mu(i,j-1),\ \mu(i-1,j-1)$ par

- Si `f[i] = g[j]` alors $\mu(i,j) = \mu(i-1,j-1)$
- Sinon si $\lambda(f_{i-1}, g_j) > \lambda(f_i, g_{j-1}$ alors $\mu(i,j) = \mu(i-1,j)$
- Sinon $\mu(i,j) = \mu(i,j-1)$
- Noter que pour $i = p$ on a $\mu(p,j) = j$.

Question 7. Donner un algorithme de calcul de plus long sous-mot commun fonctionnant en temps $O(mn)$ et en espace $O(min(n,m))$.

On procède récursivement par la découpe de f en deux, puis la détermination de k, recherche des plus longs sous mots communs à f_p, g_k puis f''_p, g''_k et concaténation des deux mots trouvés :

```
static int[] maxSmot(int[] f, int n1, int n2, int[] g, int m1, int m2){

    if ((n1 > n2) || (m1 > m2)) return null;
    else if (n1 == n2)
        if (occur(f[n1], g, m1, m2)) {
            int[] res = new int[2]; res[1] = f[n1];
            return res;}
        else return null;
    else if (m1 == m2)
            if (occur(g[m1], f, n1, n2)) {
            int[] res = new int[2]; res[1] = g[m1];
    return res;  }
         else  return null;
    else{
         int p = (int) (n1 + n2) /2;
        int k = decompose (f, n1, n2, g, m1, m2);
            int[] u = maxSmot(f, n1, p, g, m1, k);
            int[] v = maxSmot(f, p + 1, n2, g, k + 1, m2);
            return concat(u,v);}
    }
```

La valeur de k s'obtient par

```
static int decompose  (int[] f, int n1, int n2, int[] g, int m1, int m2){
// calcul de lambda et mu pour determiner ou il faut couper g
// lorsqu'on découpe f en deux facteurs egaux;

    int p = (int) (n1 + n2)/2;
    int n = (1 + n2 - n1);
    int m = (1 + m2 - m1);
    int[][] lambda = new int[2][m+1];
    int[][] mu = new int[2][m+1];
    lambda[0] = maxLong(f, n1, p, g, m1, m2);
    for (int j = 0; j <= m; ++j) mu[0][j] = j;
```

```
    for (int i = p + 1; i <= n2 ; ++i){
        for (int j = 1; j <= m ; ++j)
            if (f[i] == g[j + m1 -1]) {
                lambda[1][j] = lambda[0][j-1]+1;
                mu[1][j] = mu[0][j-1];}
            else if (lambda[0][j] > lambda[1][j-1]){
                 lambda[1][j] = lambda[0][j];
                 mu[1][j] = mu[0][j];}
            else{
                lambda[1][j] = lambda[1][j-1];
                mu[1][j] = mu[1][j-1];}
         for (int j = 1; j <= m ; ++j) {
               lambda[0][j] = lambda[1][j];
               mu[0][j] = mu[1][j];}
     }
     int res = mu[0][m]+ m1 - 1;
     return (res);
}
```