TD de Algorithmique n° 7 : Recherche exactement gène

17 Novembre 2011
(Correction)

I) Le Problème

Etant donné une séquence s de taille n et un motif g de taille m, trouver si et où g apparait dans s.

On prendra l'exemple de : Trouve t'on `AGGTAGTC` dans
`CCGTATGAGGTTAGTCAGGTACTAGGTAGGTAGTCTAACTTGCAGGTAGGC` ?

II) Naivement

Exercice 1 :

1. Ecrire un algorithme qui résoud ce problème
2. Quel est son coût dans le pire cas.

Correction :

```
Trouver(s,g)
1  i ← 0 ; j ← 0
2  while (s[i] ≠ \0&& g[j] ≠ \0)
         do
3           if (s[i]=g[j])
4              then i++ ; j++
5              else  i ← i − j +1 ; j ← 0
6  if (g[j] ≠ \0)
7     then return i − j
8     else  return −1
```

III) Ne pas revenir sur ses pas (KMP)

On cherche maintenant à ne parcourir qu'une seule fois la séquence. Pour cela, il faut déterminer la position d'où l'on doit recommencer la recherche en cas d'échec pour toute position du gène.

Par exemple, si une lettre autre que `T` apparait après la lecture de `AGGTAG`, on peut, sans reculer, regarder s'il est possible de repartir du motif `AG`. Si c'est un `T`, la lecture reprend là. Sinon, on sait que l'on recherche le gène depuis le début à partir de la position actuelle.

Un *préfixe* de t est un mot constitué des i première lettre de t. Un *suffixe* des i dernières. Ils sont dit *stricts* si ils sont strictement plus court.

Exercice 2 :

La première étape est donc de calculer la taille du plus long préfixe strict qui est aussi suffixe de tous les préfixes du gène (on parle de bord). Comment faire incrémentalement ?

Dans notre exemple, ce tableau est $[-1; 0; 0; 0; 0; 1; 2; 0]$. (le -1 est là pour les besoins techniques de l'algorithme final)

Pierre Boutillier <pierre(dot)boutillier_AT_pps(dot)jussieu(dot)fr>

Correction :

```
BORD(g)
1  pos ← 2 ; b ← 0
2  while (g[pos] ≠ \0)
3        do if (g[pos −1] = g[b])
4              then b++ ; T[pos] ← b ; pos++
5            elseif (b > 0)
6              then b ← T[b]
7              else  T[pos] ← 0 ;pos++
8  return T
```

Il suffit ensuite de parcourir la séquence et retournant au bord du mot en cas de non correspondance.

Exercice 3 :

Ecrire l'algorithne et évaluer sa complexité.

Correction :

```
KMP(s,g)
 1  i ← 0 ; j ← 0
 2  while (s[i + j] ≠ \0&&g[j] ≠ \0)
 3        do if (s[i + j] = g[j])
 4              then j ++
 5              else  i ← i + j −T[j]
 6                    if (T[j] > −1)
 7                       then j ← T[{j]
 8                       else  j ← 0
 9  if (g[j] ≠ \0)
10      then return i
11      else  return -1
```

IV) Ensuite

A la fin de ce TD surement trop court, on reviendra sur le TD à propos des listes en réécrivant le pseudo code "plus impérativement".