



Master BioInformatique

ANNÉE : 2012/2013

SESSION DE AVRIL 2013

PARCOURS : Master 1

UE J1BS8203 : Méthodes et outils pour la biologie des systèmes

Épreuve : Examen

Date : Lundi 8 avril 2013

Heure : 10 heures

Durée : 2 heures

Documents : autorisés

Épreuve de M. Alain GRIFFAULT

Code d'anonymat :

Avertissement

- La plupart des questions sont indépendantes.
- L'espace laissé pour les réponses est suffisant (sauf si vous utilisez ces feuilles comme brouillon, ce qui est fortement déconseillé).
- Solutions en langage algorithmique ou en Python.

Question	Points	Score
Graphes pondérés	6	
Plus longue sous-séquence commune	9	
Parcours en largeur	5	
Total:	20	

Exercice 1: Graphes pondérés

(6 points)

- (a) i. (2 points) Soit T un arbre couvrant minimal d'un graphe $G = (S, A, w)$, et soit S' un sous ensemble de S . Soit T' le sous-graphe de T induit par S' (c'est une forêt), et soit G' le sous-graphe de G induit par S' .

Montrez que si T' est connexe (donc un arbre), alors T' est un arbre couvrant minimal de G' .

- ii. (2 points) Soit T un arbre couvrant minimal d'un graphe $G = (S, A, w)$. Soit $G' = (S', A', w)$ le graphe obtenu à partir de G en ajoutant un nouveau sommet s' et ses arcs incidents.

Montrez que T' obtenu en ajoutant à T le sommet s' et un de ses arcs incidents de poids minimal n'est pas toujours un arbre couvrant minimal de G' .

- (b) (2 points) Donnez un exemple d'un graphe orienté pondéré $G = (S, A)$, de fonction de pondération $w : A \rightarrow \mathbb{N}$ et d'origine s , tel que G satisfasse la propriété suivante : *Pour tout arc $(u, v) \in A$, il existe une arborescence des plus courts chemins de racine s qui contient (u, v) et une autre qui ne contient pas (u, v) .*

Exercice 2: Plus longue sous-séquence commune

(9 points)

Soit w un mot. Les mots u obtenus en retirant un nombre quelconque (entre 0 et $\text{len}(w)$) de lettres forment les sous-séquences du mot w . Exemple : si $w = abacb$, alors

$$\begin{aligned} \text{sseqs}(w) = \{ & \epsilon, a, b, c, ab, aa, ac, ba, bc, bb, cb, aba, abc, abb, aac, aab, acb, bac, bab, bcb, \\ & abac, abab, abcb, aacb, bacb, abacb \} \end{aligned}$$

Soit w_1 et w_2 deux mots. Il est possible de calculer $\text{sseqs}(w_1) \cap \text{sseqs}(w_2)$, donc de calculer la longueur de la plus longue sous-séquence commune à ces deux mots.

Le problème de la *Plus longue sous-séquence commune* (PLSC) consiste à trouver **une** sous-séquence commune de longueur maximale.

Notations : Soit w un mot. On note $w[i]$ la $(i+1)^{\text{eme}}$ lettre de w , et w_i le mot composé des i premières lettres du mot w . Par convention w_0 désigne le mot vide ϵ .

Propriété : Soient $u = u_{m+1}$ et $v = v_{n+1}$ deux mots, et soit $w = w_{k+1}$ une PLSC de u et v , alors

$$\begin{cases} \text{si } u[m] = v[n] & \text{alors } w[k] = u[m] \text{ et } w_k \text{ est une PLSC de } u_m \text{ et } v_n \\ \text{si } u[m] \neq v[n] & \text{alors } w[k] \neq u[m] \Rightarrow (w_{k+1} \text{ est une PLSC de } u_m \text{ et } v_{n+1}) \\ \text{si } u[m] \neq v[n] & \text{alors } w[k] \neq v[n] \Rightarrow (w_{k+1} \text{ est une PLSC de } u_{m+1} \text{ et } v_n) \end{cases}$$

La propriété précédente permet l'écriture d'algorithmes basés sur la programmation dite *dynamique*, technique qui utilise des tableaux de stockage d'informations pour éviter de répéter des calculs. Pour calculer $\text{PLSC}(u, v)$, une matrice C va contenir pour chaque couple d'indice (i, j) la longueur de $\text{PLSC}(u_i, v_j)$. Cette matrice peut se calculer ainsi :

$$c[i, j] = \begin{cases} 0 & \text{si } i = 0 \text{ ou } j = 0 \\ c[i-1, j-1] + 1 & \text{si } i > 0, j > 0 \text{ et } u[i-1] = v[j-1] \\ \max(c[i, j-1], c[i-1, j]) & \text{si } i > 0, j > 0 \text{ et } u[i-1] \neq v[j-1] \end{cases}$$

- (a) Soit le programme python suivant :

```
def plscCodage(u, v):
    # Initialisation de la matrice avec des 0
    code = [0]*(len(u)+1)
    for i in range(len(code)):
        code[i] = [0]*(len(v)+1)
    # Calcul du codage
    for i in range(1, len(u)+1):
        for j in range(1, len(v)+1):
            if u[i-1]==v[j-1]:
                code[i][j] = code[i-1][j-1]+1
            else:
                code[i][j] = max(code[i][j-1], code[i-1][j])
    return code
```

- i. (2 points) Remplissez le tableau suivant en exécutant l'appel `plscCodage('abcbdad', 'bdcaba')`.

		0	1	2	3	4	5	6
		ε	b	d	c	a	b	a
0	ε							
1	a							
2	b							
3	c							
4	b							
5	d							
6	a							
7	b							

- ii. (1 point) Donnez et justifiez la complexité du programme `plscCodage`.
- (b) i. (4 points) Donnez un algorithme ou bien un programme python `plscDecodage(u,v,code)` qui retourne une des plus longues sous-séquences communes à `u` et `v` en utilisant le tableau `code`, qui est le résultat de l'appel `plscCodage(u,v)`

- ii. (1 point) Donnez le résultat de la suite d'instructions :

```
code = plscCodage(u,v)
plsc = plscDecodage(u,v,code)
print(plsc)
```

- iii. (1 point) Donnez et justifiez la complexité de votre programme `plscDecodage`.

Exercice 3: Parcours en largeur**(5 points)**

Définition : Un graphe biparti est un graphe non orienté $G(S, A)$ dans lequel S peut être partitionné en deux ensembles S_1 et S_2 tels que $(u, v) \in A$ implique soit $(u, v) \in S_1 \times S_2$, soit $(u, v) \in S_2 \times S_1$.

En d'autres termes, toutes les arêtes passent d'un ensemble à l'autre.

Propriété : Un graphe non orienté $G(S, A)$ est connexe si pour n'importe quel sommet $s \in S$, l'exécution de l'algorithme `ParcoursLargeur(G, s)` vu en cours colorie tous les sommets en noir.

En d'autres termes, tous les sommets sont accessibles depuis n'importe quelle racine.

- (a) (4 points) Donnez un algorithme (ou un programme python) basé sur le parcours en largeur vu en cours qui retourne **Vrai** si et seulement si le graphe G est biparti et connexe.

- (b) (1 point) Donnez et justifiez la complexité de votre algorithme.