M1 informatique 2023-2024

Examen d'algorithmique
Mardi 9 janvier 2024 8h30 – 11h / Aucun document autorisé

Mode d'emploi : Le barème est donné à titre indicatif. **La qualité de la rédaction des algorithmes et des explications sera très fortement prise en compte pour la note.** On peut toujours supposer une question résolue et passer à la suite.

1 Diviser-pour-régner (4 points)

On considère l'algorithme ci-dessous :

```
Def P(Liste L[1..n]) :
Si L est vide Alors Retourner [ ]
Si L est [x] Alors Retourner [x]
Si L est [x;y]  Alors Retourner [y;x]
Sinon
  L1 = P(L[1.. n/3])
  L2 = P(L[1+n/3..2n/3])
  L3 = P(L[1+2n/3..n])
  Retourner L3:L2:L1
```

NB : le symbole ":" représente la concaténation de deux listes. On écrit "`L est [x]`" lorsque `L` est composé d'un unique élément dont la valeur est indiquée par la variable `x`, et de manière identique "`[x;y]`" indique qu'il y a deux valeurs `x` et `y`... Enfin "`L[i..j]`" représente une copie de la sous-liste d'indices `i` à `j` (inclus).

1. Décrire ce que fait l'algorithme `P` appelé sur la liste $[4; 1; 8; 7; 5; 3; 2; 9; 14; 17; 6]$
2. Que fait l'algorithme `P` ?
3. Quelle est sa complexité si on suppose que les opérations de découpage et de concaténation prennent un temps constant ?
4. Quelle est sa complexité si on suppose que les opérations de découpage et de concaténation prennent un temps linéaire dans la taille de L ?
5. Quelle est sa complexité si on suppose que les opérations de découpage et de concaténation prennent un temps logarithmique dans la taille de L ?

2 Plus long sous-tableau stable (6 points)

Les questions sont toutes indépendantes.

Étant donné un tableau d'entiers `T` d'indices $0, \ldots, |\texttt{T}|-1$, on cherche à calculer la taille du sous-tableau stable le plus long de `T`. Un sous-tableau stable est une suite continue d'indices du tableau contenant la même valeur. Par exemple, avec le tableau `T = [1,6,8,8,8,4,2,1,1,8,4,4]`, la valeur recherchée est 3 et elle correspond au sous-tableau d'indices 2 à 4, donc de taille 3.

1. Proposer un algorithme `algo1`$(\texttt{T}, g, d)$ de type Diviser-pour-régner qui renverra la taille du sous-tableau stable le plus long entre les indices g et d (inclus) de `T`.
   Indication : on coupera la zone $g \ldots d$ en deux.
   Donner sa complexité.
   Appliquer votre algorithme sur `T = [1,6,8,8,8,4,2,1,1,8,4,4]`
2. On cherche à présent des algorithmes de programmation dynamique...
   (a) Proposer un algorithme pour construire des tableaux `Nb[-]` et `Nbf[-]` tels que :
   — `Nb[i]` est la taille d'un sous-tableau stable le plus long dans la partie de `T` restreinte aux indices $0, \ldots, i$, et
   — `Nbf[i]` est la taille d'un sous-tableau stable le plus long dans la partie de `T` restreinte aux indices $0, \ldots, i$ et se terminant à l'indice i.
   Donner sa complexité. Appliquer le sur `T = [1,6,8,8,8,4,2,1,1,8,4,4]`
   (b) Reprendre l'algorithme de la question 2a en évitant l'utilisation des tableaux. Quelle est la complexité de cet algorithme ?
   (c) Ajouter un moyen de renvoyer non seulement la longueur d'un sous-tableau stable de longueur maximale mais aussi l'indice où il commence (c'est-à-dire sa position dans le tableau).

3 Ordonnancement (4 points)

On dispose d'un ordinateur muni d'un unique processeur. Il faut exécuter n tâches $t_1, \ldots, t_n$ sur le processeur et pour cela décider de l'ordre d'exécution de ces tâches. **On suppose ici qu'il est possible d'exécuter les n tâches.**

La tâche t_i se définit par une paire d'entiers (d_i, dl_i) où d_i est la durée d'exécution de la tâche sur le processeur et dl_i est sa deadline, c'est-à-dire la date (un entier positif ou nul) à laquelle la tâche i doit être terminée. Il faut bien sûr que d_i soit inférieure ou égale à dl_i, et en cas d'égalité $d_i = dl_i$, cela signifie que la tâche i doit impérativement être exécutée en premier (au temps 0) pour qu'elle soit achevée au temps dl_i.

Par exemple, si $n = 3$ et $t_1 = (2, 5)$, $t_2 = (10, 18)$ et $t_3 = (3, 6)$, alors on peut choisir la séquence d'exécution $t_1t_3t_2$ (on commence t_1 au temps 0, puis t_3 au temps 2 et finalement t_2 au temps 5) ou la séquence $t_3t_1t_2$ (t_3 au temps 0, t_1 au temps 3 et t_2 au temps 5), mais il n'est pas possible de choisir $t_2t_1t_3$ (car cela implique de commencer la tâche t_1 au temps 10 qui est supérieur à sa deadline).

1. Montrer que prendre les tâches dans l'ordre des durées croissantes n'est pas un algorithme correct.
2. Proposer un algorithme pour résoudre le problème. Justifier votre algorithme. Donner sa complexité.

3. Si les tâches sont désormais des triplets (s_i, d_i, dl_i) où s_i est la date minimale à partir de laquelle la tâche peut être exécutée (et d_i est la durée, et dl_i la deadline).
   Par exemple : si $t_1 = (0, 2, 5)$, $t_2 = (4, 10, 18)$ et $t_3 = (1, 3, 6)$, alors la seule séquence possible est $t_1 t_3 t_2$: on exécute t_1 au temps 0 et jusqu'à 2, puis on exécute t_3 du temps 2 (compatible avec $s_3 = 1$) au temps 5 (compatible avec $dl_3 = 6$) puis on exécute t_2 du temps 5 au temps 15 (compatible avec dl_2).
   (a) Est-ce que l'algorithme de la question 2 est toujours correct ? (justifier)
   (b) Est-il parfois nécessaire d'attendre sans exécuter aucune tâche disponible ?
   (c) Une idée d'algorithme ?

4 Comparaisons de séquences - 6 points

On s'intéresse ici à la comparaison de deux séquences d'ADN, c'est-à-dire des mots sur l'alphabet $\Sigma = \{\texttt{A,C,G,T}\}$. On considère donc deux mots u et v sur cet alphabet (les deux mots peuvent avoir des tailles différentes).

L'objectif de cet exercice est de trouver le (ou un) meilleur alignement possible de u et v. Un **alignement** de u et v est une paire (U, V) où U et V sont des mots de même longueur (appelée la taille de l'alignement), U correspond au mot u où des espaces (`'_'`) ont éventuellement été ajoutés à certaines positions et V correspond au mot v où des espaces (`'_'`) ont éventuellement été ajoutés à certaines positions (donc $k \geq |u|$ et $k \geq |v|$). On interdit juste d'insérer des espaces aux mêmes positions, c'est-à-dire que dans U et V, il ne doit pas y avoir une position i telle que $U[i] = V[i] = \texttt{'\_'}$.

On définit une fonction score `score` qui associe un entier à un alignement. La fonction `score` d'un alignement (U, V) de taille k est définie comme la somme du score de chaque position i défini par : -2 si $U[i]$ ou $V[i]$ est un espace, 1 si $U[i]$ et $V[i]$ contiennent la même lettre de $\{\texttt{A,C,G,T}\}$, et -1 sinon (quand il y a deux lettres différentes). Formellement, on a donc :

$$\texttt{score}(U, V) = \sum_{i=1}^{k} \texttt{cmp}(U[i], V[i]) \qquad \texttt{cmp}(a, b) = \begin{cases} -2 & \text{si } a = \texttt{'\_'} \text{ ou } b = \texttt{'\_'} \\ +1 & \text{si } a = b \text{ et } a \in \{\texttt{A,C,G,T}\} \\ -1 & \text{si } a \neq b \text{ et } a, b \in \{\texttt{A,C,G,T}\} \end{cases}$$

Plus le score est grand, plus l'alignement des séquences est bon (et moins il y a de différences entre les séquences ainsi positionnées). On cherche donc à trouver des alignements qui maximisent leur score.

Exemple. Par exemple, si `u=GATACCA'` et `v='GAACGTCCT'`, alors on peut considérer les deux alignements suivants (avec leur score) :

U : `GATA__CCA`	U : `GATA___CCA`
V : `GAACGTCCT`	V : `GA_ACGTCCT`
score : -3	score : -4

Pour le premier, le score -3 vient de $1 + 1 - 1 - 1 - 2 - 2 + 1 + 1 - 1$.

Mais les chaînes ci-dessous **ne sont pas** des alignements :

U : `GATAC_CA`	U : `GA_TA___CCA`
V : `GAACGTCCT`	V : `GA__ACGTCCT`
Problème de longueur !	Deux espaces à la position 3 !

1. Soient `u='AGTTAA'` et `v='ACGTT'`. Proposer deux alignements pour u et v (essayer d'en trouver au moins un bon !) et indiquer leurs scores respectifs.
2. On souhaite calculer le **score** maximal possible pour un alignement de deux chaînes $u = u_1 \dots u_m$ et $v = v_1 \dots v_n$. On va construire une matrice S de taille $(m+1) \times (n+1)$, telle que $S[i, j]$ correspondra au score maximal possible pour un alignement du mot $u_1 \dots u_i$ et du mot $v_1 \dots v_j$ (donc deux préfixes des mots u et v) pour $0 \leq i \leq m$ et $0 \leq j \leq n$. La valeur $S[0, j]$ (correspondra au score d'un alignement de la chaîne vide avec le mot $v_1 \dots v_j$ (et de même pour $S[i, 0]$).

   (a) Compléter le tableau ci-dessous pour les chaînes $u =$ `GATA` et $v =$ `GTA`.

S	0	1	2	3
0	.	.	.	.
1	.	.	.	.
2	.	.	.	.
3	.	.	.	.
4	.	.	.	.

   (b) Donner la définition de $S[0, j]$ et $S[i, 0]$.

   (c) Donner la définition de $S[i, j]$ en fonction de $S[i-1, j]$, $S[i-1, j-1]$ et $S[i, j-1]$ (et de u_i et v_j).

   (d) Donner un algorithme pour calculer le tableau S. Quelle est sa complexité ?

3. Etant donné deux mots u et v et le tableau S donnant les scores maximaux de leur préfixes, proposer un algorithme qui renvoie un alignement optimal pour u et v. Cet alignement pourra être codé sous la forme d'une pile de paires de lettres de `{A,C,G,T,_}` (une paire par position). Par exemple, la pile `(A,A),(G,T),(T,T),(A,_),(A,A)` représente l'alignement `(AGTAA,ATT_A)`.

Annexe

On rappelle que le Master theorem permet d'évaluer les fonctions de la forme $t(n) = a \cdot t(\frac{n}{b}) + f(n)$ qui apparaissent dans les algorithmes diviser-pour-régner, et où a et b sont des rationnels tels que $a \geq 1$ et $b > 1$, et où $f(n)$ est une fonction positive. Ici on considère la **version simplifiée** (vue en cours) lorsque $f = \Theta(n^\alpha)$ (par ex. lorsque f est un polynôme de degré α). On distingue trois cas :

1. Si $\alpha < \log_b(a)$, on a : $t(n) = \Theta(n^{\log_b a})$.
2. Si $\alpha = \log_b a$, on a : $t(n) = \Theta(n^{\log_b a} \cdot \log n)$.
3. Si $\alpha > \log_b(a)$, alors $t(n) = \Theta(n^\alpha)$.

NB : $\frac{n}{b}$ peut aussi désigner $\lceil \frac{n}{b} \rceil$ ou $\lfloor \frac{n}{b} \rfloor$.