MPCI

Programmation et Algorithmes

Examen Terminal

Vendredi 13 mai 2022

On rappelle qu'aucun document, ni équipement électrique ou électronique n'est autorisé.

Lorsque l'on vous demande d'écrire de décrire ou de donner un algorithme cela signifiera toujours en donner un pseudo-code

1 Chaines de caractères

On rappelle qu'une chaine de caractères est une suite finie $a = a_0 \dots a_{n-1}$ d'un alphabet fini $\mathcal{A}$. Une suite $b = b_0 \dots b_{m-1}$ est une *sous-suite* de a s'il existe $0 \leq i < n$ tel que $a_{i+j} = b_j$ pour tout j allant de 0 à $m - 1$.

1.1 Algorithme naïf

1. donnez l'algorithme *naïf* (c'est à dire qu'il reproduit exactement la définition) permettant de savoir si b est une sous-chaine de a ou non.

2. donnez et justifiez la complexité maximale et minimale de cet algorithme.

3. rappelez et donnez un argument de justification concernant sa complexité en moyenne lorsque b n'est pas une sous-chaine de a.

1.2 Nombre d'occurrences

1. modifiez l'algorithme de la question 1.1 pour qu'il rende le nombre de fois où b est une sous-chaine de a. Vous justifierez votre algorithme.

2. donnez et justifiez la complexité maximale et minimale de cet algorithme.

1.3 Modification

L'algorithme de Boyer-Moore-Horspool est une modification de l'algorithme de recherche naïf. Il fonctionne de la façon suivante :

- pour un indice i donné j décroit de $m - 1$ à 0.

- lors de la première non correspondance ($a_{i+j} \neq b_j$ et $a_{i+j'} = b_{j'}$ pour $j < j' < m$), l'indice i est décalé :
  - de m cases si a_{i+m-1} n'est pas un caractère de b.
  - de $m - k - 1$ cases où k est l'indice le plus grand dans b du caractère a_{i+m-1}.

1. démontrez que ce fonctionnement est bien une façon valide de chercher si b est une sous-chaine de a.
2. proposez une structure permettant de connaître le décalage à effectuer en $\mathcal{O}(1)$ opérations en moyenne (*indication* : vous pourrez utiliser un dictionnaire), puis utilisez la pour écrire un algorithme permettant de calculer le décalage. Vous en donnerez la complexité.
3. proposez une modification de l'algorithme de la question 1.2 qui utilise la méthode de Boyer-Moore-Horspool pour la recherche.
4. donnez et justifiez la complexité maximale et minimale de cet algorithme.

2 Algorithme glouton

On appelle *équilibrage de charge* le problème suivant :

- on possède m machines et n tâches à effectuer.
- chaque tâche j nécessite t_j unités de temps pour être effectuée par une machine.
- pour chaque machine i, on associe l'ensemble M_i des tâches effectuées par celles-ci, et on note T_i le temps passé à effectuer ses tâches : $T_i = \sum_{j \in M_i} t_j$.

On cherche à trouver les ensembles M_i permettant de minimiser la quantité : $\max_{1 \leq i \leq m} T_i$. On note $T^\star$ ce minimum.

2.1 Quelques propriétés

1. montrez que l'on a $T^\star \geq \max_{1 \leq j \leq n} t_j$.
2. montrez que l'on a $T^\star \geq \frac{1}{m} \sum_{1 \leq j \leq n} t_j$ (**attention**, c'est bien $\frac{1}{m}$ et non $\frac{1}{n}$).

2.2 Un algorithme glouton

1. Proposez un algorithme glouton permettant de résoudre le problème. Cet algorithme glouton ajoutera itérativement une tâche à la machine i réalisant $T_i = \min_{1 \leq j \leq m} T_j$.
2. dans quel ordre proposez vous de ranger les tâches ? Justifiez votre réponse.
3. montrez que s'il y a m tâches ou moins à classer, l'algorithme glouton trouve la solution optimale.

2.3 Propriétés

On considère une réalisation de l'algorithme. Soit $i^\star$ la machine réalisant $T_{i^\star} = \max_{1 \leq i \leq m} T_i$ à la fin de l'algorithme, et j l'indice de la dernière tâche qui lui a été assignée au cours de l'exécution de l'algorithme.

1. montrez qu'à la fin de l'algorithme, on a $T_{i^\star} - t_j \leq T_k$ pour tout k.
2. en déduire que $T_{i^\star} - t_j \leq \frac{1}{m} \sum_{1 \leq k \leq m} T_k$.
3. déduire de la déduction que $T_{i^\star} - t_j \leq T^\star$.
4. puis que $T_{i^\star} \leq 2 \cdot T^\star$.

2.4 Performance

1. en utilisant 2.3.4, montrez que la solution proposée par l'algorithme glouton est au pire 2 fois moins bonne que la solution optimale.
2. montrer que cette performance est atteinte quelque soit l'ordre des tâches utilisé.

3 Problème

Ce problème est une adaptation d'un problème donné au phases de qualification de la coding jam de google 2022 (`https://codingcompetitions.withgoogle.com/codejam`) qui est un concours d'algorithmie et de code (comme quoi l'algorithmie sert pour trouver du travail :-)). Il n'y avait bien sur aucune indication pour résoudre le problème.

- Une chaine de caractères $a = a_0 \dots a_{n-1}$ est appelée une *tour* si pour tous $0 \leq i < j < n$ tels que $a_i = a_j$, alors $a_i = a_k$ quelque soit $i \ \leq k \leq j$.
- la concaténation + de 2 chaines $a = a_0 \dots a_{n-1}$ et $b = b_0 \dots b_{m-1}$ est la chaine $a + b = a_0 \dots a_{n-1} b_0 \dots b_{m-1}$.
- pour une chaine $a = a_0 \dots a_{n-1}$ donnée, on notera $a_{-1} = a_{n-1}$.

Le problème est le suivant : on possède un ensemble $\mathcal{T}$ de k tours, peut-on concaténer ces k tours entre elles pour former une tour ?

Par exemple :

- *aaaaabbbbbc* est une tour.
- *abba* n'est pas une tour.

- si $\mathcal{T} = \{aaaaabbbbbc, ddd, cc\}$, il existe une solution : $aaaaabbbbbc + cc + ddd = aaaaabbbbbcccddd$. Notez que toutes les concaténations ne fonctionnent pas forcément, la concaténation $cc + aaaaabbbbbc + ddd = ccaaaaabbbbbcddd$ ne donne pas une tour par exemple.
- si $\mathcal{T} = \{aaaaabbbbbc, ddd, b\}$, il n'existe pas de solution.

3.1 Vérification

Soit a une chaine de caractères. Ecrivez un algorithme qui vérifie si a est une tour ou non.
Vous prouverez la véracité et vous indiquerez la complexité de cet algorithme.

3.2 Réduction du problème

1. montrer que s'il existe une solution au problème, alors pour toute tour $a \in \mathcal{T}$ telle que $a_0 = a_{-1}$, s'il existe $b \in \mathcal{T}$ telle que $b \neq a$ et $b_0 = a_0$ (respectivement $b_{-1} = a_0$), alors il existe une solution à $(\mathcal{T} \backslash \{a, b\}) \cup \{c\}$ avec $c = a + b$ (respectivement $c = b + a$) et cette solution est aussi solution pour $\mathcal{T}$.

2. montrer que s'il existe une solution au problème, alors pour toute tour $a \in \mathcal{T}$ telle que $a_0 \neq a_{-1}$ il existe au plus une tour $b \neq a$ avec $b_0 \neq b_{-1}$ telle que $a_0 = b_{-1}$ et au plus une tour $b \neq a$ avec $b_0 \neq b_{-1}$ telle que $a_{-1} = b_0$. De plus, si ces deux tours existent, elles sont différentes.

3. montrer que s'il existe une solution au problème, alors pour toute paire de tours $a \neq b \in \mathcal{T}$ telles que $a_{-1} = b_0$ (respectivement $a_0 = b_{-1}$), alors il existe une solution à $(\mathcal{T} \backslash \{a, b\}) \cup \{c\}$ avec $c = a + b$ (respectivement $c = b + a$) et cette solution est aussi solution pour $\mathcal{T}$.

4. montrer que s'il existe une solution au problème, alors s'il existe une tour $a \in \mathcal{T}$ telle qu'il n'existe pas une autre tour $b \in \mathcal{T}$ avec $a_0 = b_{-1}$ ou $a_{-1} = b_0$ alors :
   - il existe une solution $t^\star$ à $\mathcal{T}^\star = \mathcal{T} \backslash \{a\}$.
   - $a + t^\star$ est une solution de $\mathcal{T}$.

3.3 Résolution du problème

En utilisant la partie précédente, donnez un algorithme pour résoudre le problème.