Algorithmique

Partiel du 13 novembre 2013
durée 2 heures

Le polycopié du cours est le seul document autorisé.
Les exercices sont indépendants.
Les procédures et fonctions devront être écrites en pseudo-code (pas en Java, Caml, ...)
Toutes les réponses devront être correctement justifiées.
Le chiffre en regard d'une question est une indication sur sa difficulté ou sa longueur.

1 Partitions

On définit une variante *nommée* du type abstrait `partition` comme suit :

- Donnée : une partition P de $\{1, \ldots, N\}$, chaque classe $X \in P$ étant désignée par un entier de $\{1, \ldots, M\}$, *son nom*.
  Attention : le nom d'une classe n'est pas nécessairement un élément de la classe et deux classes distinctes ne doivent pas avoir le même nom.
- Opérations :

$$\begin{aligned} \texttt{créer} &: \text{Int} \times \text{Int} \to \texttt{Partition} \\ \texttt{find} &: \texttt{Partition} \times \text{Int} \to \text{Int} \\ \texttt{union} &: \texttt{Partition} \times \text{Int} \times \text{Int} \times \text{Int} \to \texttt{Partition} \\ \texttt{renommer} &: \texttt{Partition} \times \text{Int} \times \text{Int} \to \texttt{Partition} \end{aligned}$$

- `Partition(N,M) P` crée une partition P composée des singletons de $\{1, \ldots, N\}$ et dont l'ensemble de noms est $\{1, \ldots, M\}$.
  Il faut $N \leq M$ et initialement, le nom du singleton $\{i\}$ est i.
- `P.find(i)` retourne le nom de la classe de P qui contient i.
- `P.union(x,y,z)` fusionne les classes de noms x et y en une classe de nom z.
  Ne fait rien s'il n'y a pas de classe de nom x ou pas de classe de nom y ou s'il y a déjà une classe de nom z et que $z \notin \{x, y\}$.
- `P.renommer(x,y)` change le nom de la classe x en y.
  Ne fait rien s'il n'y a pas de classe de nom x ou s'il y a déjà une classe de nom y.

[5] **a)** Décrire une structure de données concrète pour implémenter très efficacement le type abstrait `partition` et donner les algorithmes pour les 4 opérations sur ce type. Les opérations `union` et `renommer` devront s'exécuter en temps constant. L'opération `créer` devra être en temps $\mathcal{O}(N+M)$. Le coût d'une suite d'opérations comportant $n-1$ `unions` et m `find` sera en $\mathcal{O}((n+m)(1+\alpha(n)))$ où $\alpha(n)$ est l'inverse de la fonction d'Ackerman vue en cours.
Remarque : on peut bien sûr utiliser les théorèmes du cours.

On souhaite maintenant résoudre le problème des minima hors-ligne. La donnée est une suite σ d'entiers 2 à 2 distincts et d'instructions `extraire-min`, notées E dans la suite. Chaque préfixe de σ doit contenir au moins autant d'entiers que d'instructions E.
Par exemple : $\sigma = 5, 4, 8, 2, E, 7, E, 1, 6, E, E, 3$.
Chaque instruction E doit afficher le plus petit entier qui précède et qui n'a pas déjà été affiché. Pour l'exemple ci-dessus on affichera 2, 4, 1, 5.
On cherche à résoudre ce problème avec un algorithme *hors-ligne*, i.e., on peut lire *toute* la donnée σ *avant* de commencer à afficher le résultat.

[5] **b)** Donner un algorithme pour résoudre le problème des minima hors-ligne lorsque les entiers de la suite σ forment une permutation de $\{1, \ldots, n\}$. L'algorithme devra s'exécuter en temps $\mathcal{O}(n\alpha(n))$.
Indication : on pourra commencer par créer une partition dont la partie de nom $i \geq 1$ contient les entiers qui se trouvent entre le $(i-1)$-ème E et le i-ème E. Avec l'exemple ci-dessus, nous aurons les parties $X_1 = \{5, 4, 8, 2\}$, $X_2 = \{7\}$, $X_3 = \{1, 6\}$, $X_4 = \emptyset$, et $X_5 = \{3\}$.

2 Recherche de motifs

Dans ce problème, un texte t de n caractères est représenté par un tableau $t[1 \cdot\cdot n]$ de caractères. Pour $i, j \geq 0$, on note $t[i \cdot\cdot j]$ le facteur de t allant de la position i à la position j. Par convention, si $j = 0$ ou $i > n$ ou $j < i$ alors $t[i \cdot\cdot j] = \varepsilon$ est le mot vide.

Un *bord* de t est un préfixe *strict* de t qui est aussi suffixe de t : un bord est donc déterminé par un entier $0 \leq i < n$ tel que $t[1 \cdot\cdot i] = t[n - i + 1 \cdot\cdot n]$.

Le texte $t[1 \cdot\cdot n]$ étant fixé, on définit la fonction $\mathsf{lbm}\colon [1 \cdot\cdot n] \to [0 \cdot\cdot n-1]$ par $\mathsf{lbm}(k)$ est la longueur du bord maximal de $t[1 \cdot\cdot k]$. On a donc $\mathsf{lbm}(k) < k$.

On note $\mathsf{lbm}^+(k) = \{\mathsf{lbm}(k), \mathsf{lbm}^{(2)}(k), \ldots, \mathsf{lbm}^{(i)}(k) = 0\}$ l'ensemble des entiers obtenus en itérant l'application lbm jusqu'à obtenir 0.

[1] **a)** Donner les valeurs de la fonction lbm pour le texte *abaababbababaaaba*. Donner l'ensemble $\mathsf{lbm}^+(16)$.

On admet (provisoirement) que la fonction lbm satisfait

$$\forall\, 1 \leq k < n,\ \mathsf{lbm}(k+1) = \begin{cases} 0 & \text{si } t[k+1] \neq t[j+1],\ \text{pour tout } j \in \mathsf{lbm}^+(k) \\ j+1 & \text{si } j = \max\{\ell \in \mathsf{lbm}^+(k) \mid t[k+1] = t[\ell+1]\} \end{cases} \quad (1)$$

On considère le programme suivant :

```
PRE   : t tableau [1 .. n] de caractères
POST  : B[i] = lbm[i] pour tout 1 ≤ i ≤ n
B[1] ← 0; k ← 1
tant que k < n faire
   Inv1: ∀ 1 ≤ i ≤ k, B[i] = lbm(i)
   j ← B[k];
   tant que j > 0 et t[k+1] ≠ t[j+1] faire
      Inv2: j ∈ lbm⁺(k) et ∀ ℓ ∈ lbm⁺(k), j < ℓ ⟹ t[k+1] ≠ t[ℓ+1]
      j ← B[j]
   ftq
   si t[k+1] = t[j+1] alors j ← j+1 fsi
   B[k+1] ← j
   k ← k+1
ftq
```

[2] **b)** Montrer que le programme s'exécute en temps $\mathcal{O}(n)$.

[5] **c)** Prouver que `Inv1` et `Inv2` sont bien des invariants des boucles `tant que`. En déduire que le programme satisfait sa spécification.

[2] **d)** Donner un algorithme qui affiche toutes les occurrences d'un motif $x[1 \cdot\cdot p]$ contenues dans un texte $y[1 \cdot\cdot q]$. L'algorithme devra s'exécuter en temps $\mathcal{O}(p+q)$. Une occurrence de x dans y est déterminée par un entier i tel que $x[1 \cdot\cdot p] = y[i \cdot\cdot i+p-1]$.
Indication : considérer une lettre # qui n'apparaît ni dans le motif x ni dans le texte y, et exécuter le programme ci-dessus sur le texte $t = x \cdot \# \cdot y$ obtenu par concaténation du motif x, de la lettre # et du texte y.

[3] **e)** Prouver la propriété (1) de la fonction lbm (propriété admise ci-dessus).