Algorithmique

Examen du 10 novembre 2010

durée 2 heures 30

Le polycopié du cours est le seul document autorisé.

Les exercices sont indépendants.

Toutes les réponses devront être correctement justifiées. La rigueur des raisonnements, la clarté des explications, et la qualité de la présentation influeront sensiblement sur la note.

Le chiffre en regard d'une question est une indication sur sa difficulté ou sa longueur.

1 Partition du tri rapide

```
Procédure partition (t:tableau[1..n] de réels; i,j:entiers; k:entier)
PRE : 1 ≤ i < j ≤ n
POST : permute t[i..j] et retourne i≤k≤j tq t[i..k-1]≤t[k]<t[k+1..j]
Début
   g <- i; d <- i
   tant que d < j faire
      si t[d] ≤ t[j] alors
         échanger(t[g],t[d])
         g <- g+1
      fsi
      d <- d+1
   ftq
   échanger(t[g],t[j])
   k <- g
Fin
```

[4] **a)** Prouver à l'aide d'un invariant que la procédure `partition` ci-dessus est correcte.

Attention : On demande une preuve de Hoare. Vous devez donc définir un invariant pour la boucle `tant que`, insérer des assertions entre les lignes du programme, montrer que chaque triplet de Hoare $\{\varphi\}\ S\ \{\psi\}$ est correct (où φ et ψ sont les assertions avant et après l'instruction S), ...

2 Multiplication de matrices booléennes

Dans ce problème, la numérotation des lignes et colonnes d'une matrice commence à 0. Si B est une matrice $k \times n$, on note $B[i,j]$ le coefficient i,j de la matrice B (avec $0 \leq i < k$ et $0 \leq\ j < n$). Une matrice est booléenne si ses coefficients sont 0 ou 1.

Soient $k > 0$ un entier positif. On note $S^{(k)}$ la matrice booléenne $2^k \times k$ telle que pour tout $a \in \{0, \ldots, 2^{k-1}\}$ d'écriture binaire $a = \overline{a_{k-1} \cdots a_1 a_0}^2$, la ligne a de $S^{(k)}$ est $(a_0, a_1, \ldots, a_{k-1})$, i.e., $S^{(k)}[a,j] = a_j$ pour $0 \leq j < k$.

[2] **a)** Soit B une matrice $k \times n$ arbitraire. Montrer que l'on peut calculer la matrice $S^{(k)} \cdot B$ en temps $\mathcal{O}(2^k n)$.

[2] **b)** Soit A une matrice $m \times k$ *booléenne* et soit B une matrice $k \times n$ arbitraire. Montrer que l'on peut calculer la matrice $A \cdot B$ en temps $\mathcal{O}(mk + mn + 2^k n)$.

[1] **c)** Soit A une matrice $m \times (\ell k)$ *booléenne* et soit B une matrice $(\ell k) \times n$ arbitraire. Montrer que l'on peut calculer la matrice $A \cdot B$ en temps $\mathcal{O}(\ell(mk + mn + 2^k n))$.

[1] **d)** Soit A une matrice $n \times n$ *booléenne* et soit B une matrice $n \times n$ arbitraire. Montrer que l'on peut calculer la matrice $A \cdot B$ en temps $\mathcal{O}(\frac{n^3}{\log_2 n})$.

3 Ensemble

On définit un type abstrait T par :

- Donnée : un sous-ensemble X de $\{1, \ldots, N\}$.
- Opérations :

$$\begin{aligned} \texttt{créer} &: \quad \text{Int} \to \text{T} \\ \texttt{prendre} &: \quad \text{T} \to \text{T} \times \text{Int} \\ \texttt{mettre} &: \text{T} \times \text{Int} \to \text{T} \end{aligned}$$

- `T(N) X` crée le sous-ensemble $X = \emptyset$ de $\{1, \ldots, N\}$.
- `X.prendre()` supprime un élément arbitraire de X et le retourne.
  L'ensemble X doit être non vide.
- `X.mettre(i)` ajoute l'élément i à l'ensemble X.
  On *doit* avoir $i \in \{1, \ldots, N\}$ et on *peut* avoir $i \in X$.

[2] **a)** Donner une implémentation complète de ce type abstrait pour laquelle les opérations `prendre` et `mettre` sont réalisées en temps constant.
Attention à bien gérer le fait qu'on peut appeler `X.mettre(i)` alors que $i \in X$.

4 Équivalence pour les automates

Soit $\mathcal{A} = (Q, \Sigma, \delta, F)$ un automate déterministe complet avec Q un ensemble fini d'états, Σ un ensemble fini de lettres (l'alphabet), $\delta : Q \times \Sigma \to Q$ la fonction de transitions, et $F \subseteq Q$ l'ensemble des états acceptants (on ne précise pas d'état initial). La fonction δ est étendue aux mots par récurrence en posant pour tout $q \in Q$:

$$\delta(q, \varepsilon) = q \text{ (ε dénote le mot vide)} \quad \text{et} \quad \delta(q, va) = \delta(\delta(q, v), a) \text{ pour } v \in \Sigma^* \text{ et } \alpha \in \Sigma.$$

Si $q \in Q$, on note $\mathcal{L}(\mathcal{A}, q) = \{v \in \Sigma^* \mid \delta(q, v) \in F\}$ l'ensemble des mots acceptés par $\mathcal{A}$ en prenant q comme état initial. Deux états $p, q \in Q$ engendrent le même langage si $\mathcal{L}(\mathcal{A}, p) = \mathcal{L}(\mathcal{A}, q)$, on dit alors que p et q sont Nerode-équivalents.

On admettra (dans un premier temps) que deux états $p, q \in Q$ sont Nerode-équivalents si et seulement si il existe une relation d'équivalence $\sim$ sur Q telle que $p \sim q$ et pour tous $p', q' \in Q$ tels que $p' \sim q'$, on a

$$p' \in F \iff q' \in F \quad \text{et} \quad \delta(p', a) \sim \delta(q', a) \text{ pour tout } a \in \Sigma.$$

On suppose dans la suite que $Q = \{1, \ldots, n\}$. On considère l'algorithme ci-dessous qui utilise principalement une *partition* `P` de l'ensemble Q et un ensemble `E` de couples d'éléments de Q représenté par une *file*.

```
fonction testEquiv (p,q:entiers) : booléen
PRE : p,q ∈ Q et p ≠ q
Début
   Partition P.créer(n)
   File E.créer()
   E.enfiler((p,q))
   tant que E.nonVide() faire
      (x,y) <- E.défiler()
      si non (x ∈ F ⟺ y ∈ F) alors retourner Faux fsi
      x' <- P.find(x); y' <- P.find(y)
      si x' ≠ y' alors
         P.union(x',y')
         pour tout a ∈ Σ faire E.enfiler(δ(x,a),δ(y,a)) fpour
      fsi
   ftq
   retourner Vrai
Fin
```

On considère la propriété (Inv1) :

$$\forall (x,y) \in E,\ \exists v \in \Sigma^* \text{ tel que } \{x,y\} = \{\delta(p,v), \delta(q,v)\}.$$

[2] **a)** Montrer que (Inv1) est un invariant de la fonction `testEquiv`.
Remarque : Il n'est pas demandé ici de faire une preuve de Hoare.

[1] **b)** Montrer que si la fonction `testEquiv` retourne `Faux` alors p et q ne sont pas Nerode-équivalents.

[1] **c)** Montrer que la boucle `tant que` de la fonction `testEquiv` termine.

[1] **d)** Montrer que la complexité en temps de la fonction `testEquiv` est au pire *presque* linéaire, plus précisément, $\mathcal{O}(mn\alpha(n,n))$ où $m = |\Sigma|$ est la taille de l'alphabet et α est *l'inverse de la fonction d'Ackermann* définie en cours.
Remarque : la taille de $\mathcal{A}$ est au moins mn puisqu'il faut représenter la fonction δ.

On note $\sim_P$ la relation d'équivalence définie par la partition P : $x \sim_P y$ si x et y sont dans une même classe de P. On note $\sim_E$ la plus petite relation d'équivalence contenant les couples de E. On note $\sim_{P,E}$ la plus petite relation d'équivalence contenant $\sim_P$ et $\sim_E$. Par exemple, si $x_0 \sim_P x_1 \sim_E x_2 \sim_P x_3$ alors $x_0 \sim_{P,E} x_3$.

On définit la propriété (Inv2) :

$$\forall (i,j) \in Q^2 \text{ tel que } i \sim_P j, \text{ on a } \begin{cases} i \in F \iff j \in F \\ \text{et } \delta(i,a) \sim_{P,E} \delta(j,a) \text{ pour tout } a \in \Sigma \end{cases}$$

[3] **e)** Montrer que (Inv2) est un invariant de la boucle `tant que` (si on n'exécute pas l'instruction `retourner Faux`).
Remarque : Il n'est pas demandé ici de faire une preuve de Hoare.

[1] **f)** Montrer que si la fonction `testEquiv` retourne `Vrai` alors p et q sont Nerode-équivalents.

[Bonus] **g)** Montrer que la caractérisation admise en début d'exercice pour "p et q sont Nerode-équivalents" est correcte.
Question subsidiaire à traiter seulement si vous avez fait tout le reste, ce qui m'étonnerait. ;)