ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Sections d'Informatique et de Systèmes de Communication

Série d'exercices 3 5 Octobre 2009

1. Running time

Considérons les fonctions $f_i(n)$ dont les implémentations sont données ci-dessous en pseudo-code:

Call: $f_1(n)$
1: $a \leftarrow 0$
2: **for** $i = 1, \ldots, n$ **do**
3: **for** $j = 1, \ldots, i$ **do**
4: $a \leftarrow a + 1$
5: **return** a

Call: $f_2(n)$
1: $a \leftarrow 0$
2: **for** $i = 1, \ldots, n$ **do**
3: **for** $j = 1, \ldots, 2 \cdot n$ **do**
4: $a \leftarrow a + 1$
5: **for** $k = 1, \ldots, \lfloor \sqrt{n} \rfloor$ **do**
6: $a \leftarrow a + 1$
7: **return** a

Call: $f_3(n)$
1: $a \leftarrow 0$
2: **for** $i = 1, \ldots, n$ **do**
3: **for** $j = 1, \ldots, i$ **do**
4: **for** $k = j + 1, \ldots, i + j$ **do**
5: $a \leftarrow a + 1$
6: **return** a

Call: $f_4(n)$
1: $a \leftarrow 0$
2: **for** $i = 1, \ldots, n$ **do**
3: $a \leftarrow a + \ln(n)$
4: **for** $i = 1, \ldots, n$ **do**
5: **for** $j = i, \ldots, n$ **do**
6: $a \leftarrow a + n$
7: **return** a

a) Trouver des formules fermées pour $f_1(n)$, $f_2(n)$, $f_3(n)$ et $f_4(n)$ (C'est-à-dire sans $\sum$, $\prod$ et sans récursion).

b) Exprimer chacune de ces fonctions avec la notation θ, sous la forme $\theta(n^r \cdot \ln^s(n))$.

2. Elever une matrice au carré

Soit A la matrice 2×2 sur $\mathbb{R}$: $A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$

a) Trouver un algorithme qui calcule A^2 en utilisant 5 multiplications d'éléments de $\mathbb{R}$ (et autant d'additions d'éléments de $\mathbb{R}$ que nécessaire).

b) Peut-on généraliser cet algorithme récursivement à toute matrice $n \times n$ pour obtenir un algorithme $O(n^{\log_2(5)})$ (comme nous l'avons fait dans le cours pour l'algorithme de Strassen)? Pourquoi?

3. Algorithme

Supposons que nous disposons d'une machine avec une mémoire infinie qui peut effectuer les opérations suivantes:

- Multiplier deux nombres stockés dans la mémoire
- Stocker le résultat en mémoire

Pour un n fixé, nous aimerions programmer cette machine (i.e. développer un algorithme) qui calcule la valeur de a^n (si l'input a est donné) en effectuant aussi peu de multiplications que possible.

Nous supposons qu'au début, exactement une valeur est stockée en mémoire: a.

Par exemple, si $n = 3$ nous pourrions programmer la machine comme suit:

- $1^{\textbf{er}}$ **pas:** calculer $a * a$ et stocker le résultat en mémoire. Nous avons donc maintenant les valeurs de a et de a^2 stockées en mémoire.
- $2^{\textbf{ème}}$ **pas:** calculer $a * a^2$ et stocker le résultat. Nous avons maintenant les valeurs a, a^2 et a^3 en mémoire.

Ainsi, nous avons calculé a^3 en utilisant deux multiplications.

a) Trouver un algorithme qui calcule a^4 en utilisant 2 multiplications.

b) Trouver un algorithme qui calcule a^7 en utilisant 4 multiplications.

c) Trouver un algorithme qui calcule a^{15} en utilisant 5 multiplications.

d) Supposons que n est une puissance de 2. Trouver un algorithme qui calcule a^n en utilisant aussi peu de multiplications que possible. Exprimer le nombre de multiplications nécessaires en fonction de n.

e) Montrer que pour n'importe quel n il existe un algorithme qui calcule a^n en effectuant au plus $2 \cdot \lfloor \log_2(n) \rfloor$ multiplications.

4. Stacks et files d'attente

a) Supposons que nous disposons uniquement de la structure de données stack. Est-il possible de réaliser une file d'attente en utilisant deux stacks? Si oui, donner les algorithmes correspondants pour les opérations de la file d'attente.

b) Nous voulons de nouveau réaliser une file d'attente en utilisant deux stacks, mais avec la condition de plus que le temps de calcul des opérations est similaire à celui d'une implémentation classique de file d'attente.

Plus précisément, supposons que les opérations sur les stacks se font en temps $O(1)$ (i.e. ne dépendent pas de la taille du stack). Donner alors des algorithmes pour les opérations d'une file d'attente basée sur deux stacks qui ont la propriété que, si l'on commence avec une file vide, pour faire les m premières opérations de la file d'attente, on a besoin de $O(m)$ opérations du stack.

5. Permutations avec stacks et files d'attente

Considérons une machine avec les caractéristiques suivantes:

- Elle dispose d'un (seul) stack S.
- Elle peut lire un caractère de l'input et le déposer sur le stack—c'est l'opération `Push`.
- Elle peut enlever un caractère du stack et l'ajouter à l'output—`Pop`.
- Il n'y a pas d'autres opérations ni de stock supplémentaire.

Par exemple, la suite suivante d'instructions

`Push`, `Push`, `Push`, `Pop`, `Push`, `Pop`, `Pop`, `Push`, `Pop`, `Pop`

crée, si l'input $i = (1, 2, 3, 4, 5)$ est donné, l'output $(3, 4, 2, 5, 1)$.

a) Existe-t-il une suite d'instructions qui sort, pour le même input i, l'output $(1, 2, 3, 5, 4)$? Et l'output $(2, 3, 5, 4, 1)$? Ou encore $(3, 1, 2, 5, 4)$?

b) Montrer qu'il est possible d'obtenir la permutation $(p_1, p_2, \ldots, p_n)$ de $(1, 2, \ldots, n)$ s'il n'existe pas d'indices $i < j < k$ tel que $p_j < p_k < p_i$.

c) Si nous remplaçons le stack par une file d'attente, quelles permutations pouvons nous alors obtenir?