Licence 2 Informatique et Mathématiques
Algorithmique et Programmation 4

Véronique Jay, Eric Sanlaville

11 mars 2024, durée 1h30

Il est préférable de faire les 3 exercices dans l'ordre, même si de nombreuses questions sont indépendantes.
Tous documents et appareils connectés interdits.

1 Algorithme de codage en binaire

Dans cette section, on s'intéresse au codage en binaire d'un nombre entier. Pour cela, on propose l'algorithme $\mathcal{A}$ ci-dessous. Cet algorithme utilise le type vecteurBooleen. On suppose que si v est un vecteurBooleen, $l(v)$ retourne sa longueur (nombre de coordonnées) et $v[i]$ permet d'accéder (et de modifier, ou même de créer) sa $i$ème coordonnée, avec $0 \leq i < l(v)$. vecteurVide() crée un vecteur booléen vide.

Algorithme $\mathcal{A}$

```
Entrée k : entier
Sortie : vecteurBooleen
variables  n,i : entier ; v : vecteurBooleen
debut
        n ← k; i ← 0; v ← vecteurVide()
        tant que ( n ≥ 1) faire
                si (n mod 2 = 0) alors
                        v[i] ← 0
                sinon v[i] ← 1
                fin si
                n ← n div 2; i++
        fin tantque
        retourner (v)
fin
```

1.1 Finitude

Montrez que l'algorithme $\mathcal{A}$ termine. Pour cela vous utiliserez un variant. Lequel?

1.2 Correction

L'objectif de l'algorithme $\mathcal{A}$, c'est de retourner un vecteur contenant les chiffres de l'entrée k écrite en binaire. Par définition, si w est ce vecteur, on doit avoir : $k = \sum_{i=0}^{l(w)-1} w[i] * 2^i$.

On va montrer que le vecteur v résultat de l'algorithme vérifie bien cette égalité. Pour cela, on va montrer par récurrence sur i l'existence d'un invariant.

Notons n_i la valeur de la variable n au début de l'itération numéro i. Soit alors $I(i) = 2^i * n_i + \sum_{j=0}^{i-1} v[j] * 2^j$.

Montrer que la fonction I est un invariant, c'est à dire qu'elle est constante pour tout i lors du déroulement de l'algorithme. Votre preuve sera par récurrence sur i.

1.3 Complexité

Supposons que k soit une puissance de 2 : $k = 2^p$, avec p entier positif. En déduire que le nombre d'itérations de $\mathcal{A}$ est exactement égal à $p + 1$.

On admettra que si $2^{p-1} \leq k < 2^p$, alors le nombre d'itérations de l'algorithme est égal à p. En déduire que la complexité de l'algorithme est $\Theta(\log k)$.

2 Algorithme mystère

Soit l'algorithme suivant (le type vecteurBooleen est défini dans la première question) :

Algorithme $\mathcal{B}$

Entrées x : réel ; v : vecteurBooleen

Sortie : réel

variables R, u : réel ; i : entier

debut

    $R \leftarrow 1$; $u \leftarrow x$;

    pour ($i = 0 \rightarrow l(v) - 1$) faire

        si $v[i] = 1$ alors $R \leftarrow R * u$; fin si

        $u \leftarrow u * u$;

    fin pour

    retourner(R)

fin

2.1 Tests

Appliquez l'algorithme aux entrées suivantes en donnant bien l'évolution des variables R et u :

$$(3; [1, 1]) \quad (1/2; [0, 0, 1]) \quad (2; [1, 0, 1])$$

À votre avis, que fait cet algorithme ?

2.2 Finitude

Montrez simplement que l'algorithme B termine.

2.3 Complexité

On suppose dans la suite que l'algorithme $\mathcal{B}$ est correct. Donnez la complexité de l'algorithme en fonction de $l(v)$, en utilisant la notation de Landau Θ. On supposera que l'opération de multiplication de 2 réels est une opération élémentaire.

3 Algorithme de calcul de puissance

Soit l'algorithme ci-dessous.

Algorithme $\mathcal{C}$

1. Entrées x : réel ; k : entier
2. Sortie : réel
3. variables v : vecteurBooleen
4. debut
5. $v \leftarrow \mathcal{A}(k)$;
6. retourner $(\mathcal{B}(x, v))$;
7. fin

Expliquez (sans prouver sa correction) pourquoi l'algorithme $\mathcal{C}$ calcule bien x^k.
Déduire des deux sections précédentes la complexité de $\mathcal{C}$ en fonction de k.