2I003 – Interrogation longue de TD
durée : 1 heure 15min

15 novembre 2016

Soit x un entier naturel, on appelle *taille de x en bits* l'entier $t(x) = \lfloor \log_2(x+1) \rfloor$. Il correspond à la place mémoire de l'entier x en machine, c'est-à-dire son nombre de chiffres dans sa décomposition en base 2. Par exemple, on a $t(2) = 2$, $t(9) = 4$. En particulier, remarquez que si $x \leq 2^k - 1$, alors $t(x) \leq k$.

On se pose le problème suivant.

Multiplication de n entiers : étant donnés n nombres entiers naturels $(x_i)_{1 \leq i \leq n}$, chacun de taille $t(x_i) \leq d$, calculer le produit $\prod_{i=1}^{n} x_i$ de ces nombres.

PARTIE 1

Question 1.- Soient x, y deux entiers naturels. Montrer que $t(xy) \leq t(x) + t(y)$.

PARTIE 2

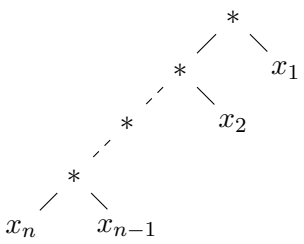
Un arbre de produits est une expression arithmétique ne comportant que des multiplications (`*`) comme opérateurs. On rappelle qu'une expression arithmétique est un arbre dont les noeuds internes sont des opérateurs et les feuilles sont des nombres.

Dans toute la suite, `X` représente une liste de taille `n`. On définit un premier algorithme de produit naïf comme suit :

```
def ProduitNaif(X, n):
    if (n == 1):
        return X[0]
    else:
        return ProduitNaif(X[1:], n-1) * X[0]
```

Question 2.1.- Écrire l'arbre de produits de l'expression arithmétique associée à `ProduitNaif(X, n)` pour `X = [3, 5, 2, 4]` et `n = 4`.

On note $T_n(X)$ l'arbre de produits suivant :



Question 2.2.- Montrer par récurrence sur n que T_n est l'arbre de produits de `ProduitNaif(X, n)`.

Question 2.3.- Donner la hauteur et le nombre de noeuds internes de T_n.

On note $M(d, e)$ la complexité de l'opération `x * y` où $t(x) = d$ et $t(y) = e$. On note aussi c_i la complexité du calcul effectué à profondeur i de l'arbre (la profondeur 0 étant la racine).

Question 2.4.- On rappelle que, par hypothèse, toutes les valeurs $x_i = \texttt{X[i - 1]}$ ont une taille $t(x_i) \leq d$. On note u_i la valeur de la clef à la profondeur $i \in \{0, \ldots, n-2\}$ après avoir réduit l'arbre (c'est-à-dire, après avoir effectué les multiplications). Par exemple, on a $u_{n-2} = x_n x_{n-1}$. Montrer que $t(u_i) \leq (n-i)d$. En déduire que $c_i \leq M((n-1-i)d, d)$.

Question 2.5.- En déduire que la complexité C_n de l'algorithme `ProduitNaif(X, n)` vérifie :

$$C_n \leq \sum_{j=1}^{n-1} M(jd, d)\,.$$

PARTIE 3

Maintenant, on s'intéresse à un algorithme de produit un peu plus évolué.

```
def ProduitEvolue(X, n):
    if (n == 1):
        return X[0]
    else:
        m = n//2
        return ProduitEvolue(X[:m], m) * ProduitEvolue(X[m:], n-m)
```

Question 3.1.- Écrire l'arbre de produits de `ProduitEvolue(X, n)` pour `X = [3, 5, 2, 4]` et `n = 4`.

On suppose à partir de maintenant que $n = 2^k$ est une puissance de 2.

Question 3.2.- Montrer par récurrence sur k que l'arbre de produits de `ProduitEvolue(X, n)` est parfait. Quelle est sa hauteur ?

Question 3.3.- Soit P un arbre parfait, et h sa hauteur. Montrer que si $i \in \{0, \ldots, h-1\}$, alors le nombre de noeuds à profondeur i de P est 2^i.

Question 3.4.- On note b_i la complexité du calcul sur un noeud interne à profondeur i, et on admet que :

$$b_i \leq M(2^{k-1-i}d, 2^{k-1-i}d)\,.$$

En déduire que la complexité B_n de l'algorithme `ProduitEvolue(X, n)` vérifie :

$$B_n \leq \sum_{j=1}^{k} 2^{j-1} M(2^{k-j}d, 2^{k-j}d)\,.$$

PARTIE 4

Question 4.1.- Dans cette question on suppose que $M(d, e) = d \times e$ (cela correspond à l'algorithme de multiplication que vous avez appris à l'école). Donner une approximation de B_n et C_n en fonction de n et d.

Question 4.2.- Dans cette question on suppose que $M(d, e) = \max(d, e)$. Donner une approximation de B_n et C_n en fonction de n et d.