2I003 – Interrogation longue de TD
Corrigé

15 novembre 2016

Soit x un entier naturel, on appelle *taille de x en bits* l'entier $t(x) = \lceil \log_2(x+1) \rceil$. 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 $t(xy) \leq t(x) + t(y)$.

Corrigé : On a :

$$\log_2(xy+1) \leq \log_2(xy+x+y+1) \leq \log_2((x+1)(y+1)) \leq \log_2(x+1) + \log_2(y+1)\,.$$

Donc, $\lceil \log_2(xy+1) \rceil \leq \lceil \log_2(x+1) + \log_2(y+1) \rceil \leq \lceil \log_2(x+1) \rceil + \lceil \log_2(y+1) \rceil$, c'est-à-dire $t(xy) \leq t(x) + t(y)$.

PARTIE 2

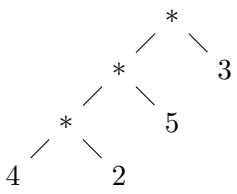
Un arbre de produits est une expression arithmétique ne comportant que des multiplications (`*`) comme opérateurs. On rappelle qu'une expression arithmétique est un arbre dont les noeuds internes sont des opérateurs et les feuilles sont des nombres.

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

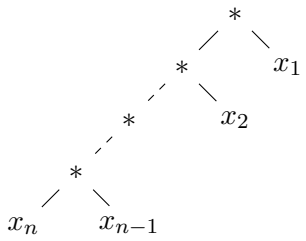
```
def ProduitNaif(X, n):
    if (n == 1):
        return X[0]
    else:
        return ProduitNaif(X[1:], n-1) * X[0]
```

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`.

Corrigé : On obtient l'arbre :

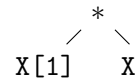


On note $T_n(X)$ l'arbre de produits suivant :

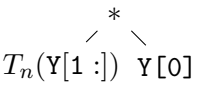


Question 2.2.- Montrer par récurrence sur n que T_n est l'arbre de produits de `ProduitNaif(X, n)`.

Corrigé :
• Base : pour $n = 2$, la propriété est vérifiée car l'arbre de produit de `ProduitNaif(X, 2)` est :



• Hérédité : Supposons que pour tout `X` de taille n, `ProduitNaif(X, n)` ait comme arbre de produit $T_n(X)$. Soit `Y` une liste d'entiers de taille $n + 1$. Alors `ProduitNaif(Y, n+1)` égal au produit `ProduitNaif(Y[1:], n) * Y[0]`. La liste `Y[1:]` a taille n, donc par hypothèse de récurrence, l'arbre de produits de `ProduitNaif(Y, n+1)` est de la forme



En développant l'arbre T_n(`Y[1 :]`), on obtient bien un arbre de la forme T_{n+1}(`Y`).

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

Corrigé : $h(T_n) = n$ et $n_i(T_n) = n - 1$.

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 $x_i =$ `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 $c_i \leq M((n-1-i)d, d)$.

Corrigé : On remarque que $u_i = \prod_{j=i+1}^{n} x_i$, donc d'après la question 1

$$t(u_i) = t\Big(\prod_{j=i+1}^{n} x_i\Big) \leq \sum_{j=i+1}^{n} t(x_i) = \sum_{j=i+1}^{n} d = (n-i)d\,.$$

Comme c_i correspond à la multiplication entre u_{i+1} et x_i, on a donc $c_i = M(t(u_{i+1}), t(x_i)) \leq M((n - i - 1)d, d)$.

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

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

Corrigé : La complexité totale est la somme des complexités, donc :

$$C_n = \sum_{i=0}^{n-2} c_i \leq \sum_{i=0}^{n-2} M((n-i-1)d, d) = \sum_{j=1}^{n-1} M(jd, d)\,,$$

en postant $j = n - i - 1$.

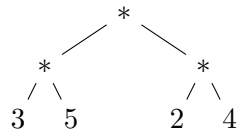
PARTIE 3

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`.

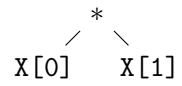
Corrigé : On obtient l'arbre :



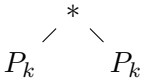
On suppose à partir de maintenant que $n = 2^k$ est une puissance de 2.

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

Corrigé :
• Base : pour $n = 2$, la propriété est vérifiée car l'arbre de produit de `ProduitEvolue(X, 2)` est :



• Hérédité : Supposons que pour tout `X` de taille 2^k, `ProduitEvolue(X, 2**k)` ait comme arbre de produit un arbre parfait. Soit `Y` une liste d'entiers de taille $n = 2^{k+1}$. Alors $m = n/2 = 2^k$, donc les appels récursifs de `ProduitEvolue(Y[m:], m)` et `ProduitEvolue(Y[:m], m)` produisent des arbres parfaits (par hypothèse de récurrence). Par ailleurs, ces arbres ont un dernier niveau rempli, car ils possèdent exactement 2^k feuilles. Pour résumer, l'arbre de produit de `ProduitEvolue(Y, n)` est de la forme



où les P_k sont parfaits avec un dernier niveau rempli. Donc il est parfait.
• Sa hauteur est $k + 1$.

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

Corrigé : Fait en TD, se montre par induction (ou par récurrence si on veut être formel).

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 :

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

Corrigé : Il y a 2^i calculs à effectuer à profondeur i, chacun avec une complexité $b_i \leq M(2^{k-1-i}d, 2^{k-1-i}d)$. Donc la complexité totale vérifie

$$B_n = \sum_{i=0}^{k-2} 2^i b_i \leq \sum_{i=0}^{k-2} 2^i M(2^{k-1-i}d, 2^{k-1-i}d) = \sum_{j=1}^{k-1} 2^{j-1} M(2^{k-j}d, 2^{k-j}d)\,.$$

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.

Corrigé : D'une part,

$$\begin{aligned} B_n &\leq \sum_{j=1}^{k-1} 2^{j-1} M(2^{k-j}d, 2^{k-j}d) \leq \sum_{j=1}^{k-1} 2^{j-1} 2^{k-j} d\, 2^{k-j} d \leq \sum_{j=1}^{k-1} 2^{2k} 2^{-j-1} d^2 \\ &\leq 2^{2k} d^2 \sum_{j=1}^{k-1} 2^{-j-1} \leq n^2 d^2 \times \frac{1}{4} \sum_{j=0}^{k-2} 2^{-j} \leq n^2 d^2 \times \frac{1}{4} \frac{1 - 2^{-k+1}}{1 - 1/2} \leq \frac{1}{2} n^2 d^2 \in \mathcal{O}(n^2)\,. \end{aligned}$$

D'autre part,

$$C_n \leq \sum_{j=1}^{n-1} M(jd, d) \leq \sum_{j=1}^{n-1} jd^2 \leq \frac{n(n-1)}{2} d^2 \leq \frac{1}{2} n^2 d^2 \in \mathcal{O}(n^2)\,.$$

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.

Corrigé : D'une part,

$$\begin{aligned} B_n &\leq \sum_{j=1}^{k-1} 2^{j-1} M(2^{k-j}d, 2^{k-j}d) \leq \sum_{j=1}^{k-1} 2^{j-1} 2^{k-j} d \leq \sum_{j=1}^{k-1} 2^{k-1} d \\ &\leq 2^{k-1}(k-1)d \leq \frac{1}{2} n \log_2(n) d \in \mathcal{O}(n \log n)\,. \end{aligned}$$

D'autre part,

$$C_n \leq \sum_{j=1}^{n-1} M(jd, d) \leq \sum_{j=1}^{n-1} jd \leq \frac{n(n-1)}{2} d \leq \frac{1}{2} n^2 d \in \mathcal{O}(n^2)\,.$$