Polytech'Lille — GIS 3 — Structures de Données — Feuille de TD numéro 3

Cette feuille de TD est à apporter lors du TP 3. Elle porte sur la notion de complexité. Le but pédagogique est le suivant : comprendre les notations asymptotiques ; comprendre comment modifier un algorithme, même compliqué, pour lui faire compter les opérations effectuées et produire une courbe expérimentale ; comprendre le principe « diviser pour régner », le type d'équation de récurrence qui lui est associé et le changement de variable classique qui permet leur résolution. On ne demande ni de comprendre toutes les finesses de l'algorithme de Karatsuba, qui sert d'exemple, ni de savoir résoudre les récurrences à la main.

1 Compréhension des notations asymptotiques

Question 1. On s'intéresse au nombre de comparaisons d'éléments effectuées par l'algorithme du tri par insertion, appliqué à un tableau de n éléments. À la question « quel est le meilleur des cas ? » un étudiant répond : « c'est le cas où $n = 0$: le tableau est vide et il n'y a aucune comparaison à effectuer ». Qu'en pensez-vous ?

Question 2. Algorithm A has a time complexity in $\Theta(n)$. Algorithm B has a time complexity in $\Theta(n^2)$. Can we claim that A is always faster than B ?

Question 3. L'algorithme A a une complexité en temps en $O(n)$. L'algorithme B a une complexité en temps en $O(n^2)$. Peut-on affirmer que A est plus rapide que B, quand n tend vers l'infini ?

Question 4. L'algorithme A a une complexité en temps en $O(n)$. L'algorithme B a une complexité en temps en $\Omega(n^2)$. Peut-on affirmer que A est plus rapide que B, quand n tend vers l'infini ?

Question 5. What about an algorithm, the complexity of which is a $\Theta(1)$?

Question 6. Un professeur démontre qu'un algorithme a une complexité en temps, dans le pire des cas, en $\Theta(n^3)$. Un étudiant intervient : « cet algorithme a donc une complexité en temps, dans tous les cas, en $O(n^3)$ ». Qu'en pensez-vous ?

Question 7. Les algorithmes A et B ont tous deux une complexité en temps en $\Theta(g(n))$. Peut-on affirmer que, quand n tend vers l'infini, les temps de calcul des deux algorithmes sont de plus en plus proches ?

2 Analyse d'algorithmes

Pour toutes les fonctions suivantes, on s'intéresse à la fonction $f(n)$ qui mesure le nombre d'opérations arithmétiques effectuées sur les doubles.

2.1 Multiplication de polynômes

Dans cette section, n désigne le nombre de coefficients de deux polynômes A et B. On les suppose de même taille pour simplifier. Ces polynômes sont donc de degré $n-1$.

2.1.1 Deux algorithmes élémentaires

Question 8. La fonction suivante calcule la somme de deux polynômes A et B de degré $n-1$. Le résultat est enregistré dans R. Que vaut $f(n)$?

```
void add_poly (double* R, double* A, double* B, int n)
{   int i;
    for (i = 0; i < n; i++)
        R[i] = A[i] + B[i];
}
```

Question 9. La fonction suivante calcule le produit de deux polynômes A et B de degré $n-1$. Le résultat est enregistré dans R. Que vaut $f(n)$?

```
void mul_poly (double* R, double* A, double* B, int n)
{   int a, b;
    for (a = 0; a < 2*n-1; a++)
        R[a] = 0.0;
    for (a = 0; a < n; a++)
        for (b = 0; b < n; b++)
            R[a+b] = R[a+b] + A[a] * B[b];
}
```

Question 10. In order to obtain a file of measures for $f(n)$, we would like to turn the procedures `add_poly` and `mul_poly` into functions which return $f(n)$. How could we proceed ?

2.1.2 La méthode d'Anatolii Alexeevitch Karatsuba

Il s'agit d'une méthode pour multiplier deux polynômes comportant n coefficients chacun (et donc de degré $n-1$), ou deux entiers de n chiffres entre eux. Le cas des entiers est un peu plus compliqué, en raison du problème du report des retenues.

Question 11. On lit une variante du texte suivant sur la *Wikipedia*. Comment le comprenez-vous ?

> *In 1960, Andrey Kolmogorov organized a seminar at the Moscow State University, where he stated the $\Omega(n^2)$ conjecture. Within a week, Karatsuba, then a 23-year-old student, found an algorithm in $\Theta(n^{\log_2(3)})$, thus disproving the conjecture. Kolmogorov was very upset about the discovery. He communicated it at the next meeting of the seminar, which was then terminated.*

Principe. On considère deux polynômes A et B comportant n coefficients (et donc de degré $n-1$) chacun. Soit $0 < p < n$ un entier. Il est possible de décomposer chacun des deux polynômes en deux parties :

```
> A := A0 + x^p * A1;
                                        p
                                A := A0 + x  A1
> B := B0 + x^p * B1;
                                        p
                                B := B0 + x  B1
```

Effectuons maintenant les opérations suivantes :

```
> R0 := A0*B0;
                                  R0 := A0 B0
> R2 := A1*B1;
                                  R2 := A1 B1

> R1 := expand ((A0+A1)*(B0+B1)-R0-R2);
                              R1 := A0 B1 + B0 A1
```

Il est possible de construire le produit $A\,B$ des deux polynômes de départ par la formule suivante :

```
> expand (R0 + x^p*R1 + x^(2*p)*R2);
                                p       p           p 2
                  A0 B0 + A0 x  B1 + x  A1 B0 + (x )  A1 B1
```

Vérification :

```
> expand (A*B);
                                p       p           p 2
                  A0 B0 + A0 x  B1 + x  A1 B0 + (x )  A1 B1
```

Supposons n pair. Posons $p = n/2$. Les polynômes A_0, A_1, B_0 et B_1 ont alors tous p coefficients (ils sont de taille p). Les polynômes R_0, R_1, R_2 sont chacun le produit de deux polynômes de taille p (donc de degré $p-1$). Ils sont donc tous les trois de degré $2\,p-2 = n-2$ et ont donc chacun $n-1$ coefficients. Leur calcul demande trois multiplications et deux additions de polynômes taille p et deux additions de polynômes de taille $n-1$. La construction du produit $A\,B$ à partir de ces trois polynômes ne demande qu'une seule addition de deux polynômes de taille $n-1$, puisque l'addition $R_0 + x^{(2\,p)}\,R_2$ revient à mettre les deux tableaux R_0 et R_2 bout-à-bout !

Question 12. En supposant que les trois multiplications sont effectuées par l'algorithme élémentaire analysé plus haut, donner une formule pour $f(n)$ et montrer que la méthode ainsi obtenue est plus efficace que la méthode élémentaire.

La méthode complète. L'idée consiste maintenant à utiliser le schéma ci-dessus pour multiplier les polynômes A_0, A_1, B_0, B_1 entre eux. En poussant l'idée dans ses retranchements, on aboutit à la fonction récursive de la figure 1. L'algorithme est simplifié pour le cas particulier où $n = 2^k$. On travaillera en TP sur une fonction mieux écrite.

Question 13. Comment transformer cette action en une fonction qui retourne $f(n)$?

Question 14. Donner une relation de récurrence avec condition initiale pour $f(n)$. On rappelle que $n = 2^k$.

Question 15 (plus difficile). Résoudre la récurrence.

```
void Karatsuba (double* R, double* A, double* B, int n)
{   double *A0, *A1, *B0, *B1, *R0, *R2;
    int p = n/2;
    double R1 [n], tmp1 [p], tmp2 [p];
    int i;

    if (n == 1)
        mul_poly (R, A, B, n);
    else
    {   A0 = A;
        A1 = A + p;
        B0 = B;
        B1 = B + p;
        R0 = R;
        R2 = R + n;
/*
 * Découper les polynômes A et B en deux se fait en temps constant
 * Les polynômes R0 et R2 sont des sous-tableaux de R.
 */
        Karatsuba (R0, A0, B0, p);
        Karatsuba (R2, A1, B1, p);
/*
 * À ce stade, R = R0 + x^(2*p)*R2
 */
        add_poly (tmp1, A0, A1, p);
        add_poly (tmp2, B0, B1, p);
        Karatsuba (R1, tmp1, tmp2, p);
        sub_poly (R1, R1, R0, 2*p-1);
        sub_poly (R1, R1, R2, 2*p-1);
/*
 * On calcule R = R + x^p*R1
 */
        R[n-1] = 0;
        for (i = 0; i < 2*p-1; i++)
            R[p+i] = R[p+i] + R1[i];
    }
}
```

FIGURE 1 – Une version simplifiée de l'algorithme de Karatsuba.