Temps de calcul des algorithmes récursifs

$\pi\lambda$

Octobre 2013
dernière compilation : 4 novembre 2015

Résumé

L'objectif de cet exercice est de retrouver numériquement les résultats du coursconcernant le temps de calcul des algorithmes *diviser pour régner* en général, et en particulier, ceux du *tri fusion* et de la *méthode de Karatsuba*.

1 Algorithme A

```
32
33 void A(int n)
34 {
35     if (n > 1) {
36         B(n);
37         A(n / 2);
38         A(n / 2);
39     }
40 }
```

La fonction récursive `A(n)` *divise* un problème de taille n en deux sous-problèmes de taille $n/2$. Elle termine par une *fusion* `B(n)`. Le temps de calcul de la fonction `A(n)` dépend uniquement de celui de la fusion `B(n)`. Dans le cas d'un tri rapide, la fusion `B` est linéaire, et le temps de calcul de `A` est de la forme $n\log(n)$. D'une manière générale, une analyse de l'arbre d'exécution de `A` donne sur le champ la formule :

$$T_A(n) = \sum_{k=0}^{l-1} 2^k \times T_B(\frac{n}{2^k}) + T_A(1) \times n + \mathrm{O}(1) \tag{1}$$

pour $n = 2^l$.

2 Mesure des temps

Nous avons vu comment mesurer les temps de calcul avec la commande du `/usr/bin/time`, dans cette expérience, nous utiliserons la fonction `clock` de la `glibc`.

```
42 void chronos(int max)
43 {
44     clock_t  s;
45     int c,  n = 1;
46     while (n < max) {
47         s  = clock();
48         A(n);
49         c  = clock() - s;
50         printf ("\n%d %d", n, c);
51         fflush (stdout);
52         n = n * 2;
53     }
54 }
```

3 Ligne de commande

La fonction `getopt()` est utilisée pour gérer les options courtes passées par la ligne de commande :

```
56 int main(int argc, char *argv[])
57 {
58     int opt,  m = 0;
59     while ((opt = getopt(argc, argv, "clns:")) != -1) {
60         switch (opt) {
61         case 'c':
62             flag  = 0;
63             break;
64         case 'l ':
65             flag  = 1;
66             break;
67         case 'n':
68             flag  = 2;
69             break;
70         case 's':
71             m = 1 << atoi(optarg);
72             break;
73         default:                    /* '?' */
74             fprintf (stderr,  "Usage: %s [cln] [-s size]\n", argv[0]) ;
75             exit (EXIT_FAILURE);
```

TAB. 1 – Les formes des temps de calcul.

T_B	T_A	facteur caché
constant	linéaire	...
logarithmique	...	...
linéaire	$n\log(n)$	...
quadratique	...	...

```
76          }
77      }
78      chronos(m);
79      return 0;
80  }
```

4 Mesure

Il s'agit d'utiliser `gnuplot` pour mettre en évidence le temps de calcul de l'algorithme A dans le cas d'une fusion à temps constant, logarithmique, linéaire et quadratique.

```
abc> wget http://langevin.univ-tln.fr/cours/PAA/tps/abc/
    abc.c
```

Vous devez notamment vous convaincre graphiquement que lorsque B est logarithmique, A n'est pas $\Theta(n\log n)$ mais linéaire.

1. Faire des mesures de temps pertinentes.
2. Utiliser `gnuplot` pour déterminer les formes des temps de calcul.
3. Compléter le tableau TAB. [1].

5 Méthode de Karatsuba

1. Implanter un algorithme A_3 à fusion linéaire.
2. Mettre en évidence le temps de calcul de la méthode de Karatsuba.

6 Tri fusion

1. Implanter une fonction `float* aleas(int n)` qui génére un tableau de n flottants aléatoires.
2. Implanter une fonction `boole test(float* t, int n)` qui retourne vrai si et seulement le tableau t de taille n est en ordre croissant.
3. Implanter le tri fusion du cours pour trier des tables de flottants.

4. Procéder aux tests et mesures de temps de calcul.
5. Déterminer la bonne valeur de *seuil* pour arrêter la récursion au profit d'un tri par selection.

7 Tri rapide

1. Implanter le tri rapide à division de Lomuto.
2. Procéder aux tests.
3. Faire une expérience numérique pour mettre en évidence le nombre moyen de comparaisons.