Preuve et Analyse des Algorithmes

12 juin 2013

1. Temps de calcul. Une implantation d'un algorithme de temps de calcul $n\sqrt{n}$ traite une instance de taille $n = 64$ en 8 micro-secondes.

1. Quel sera le temps de calcul pour une instance de taille 512 ?
2. Donner un exemple d'algorithme ayant cette complexité.

2. Comparaison des algorithmes de tris. Pour un entier n, et algorithme de tri X, on note :

$$t_X(n) = \frac{\bar{T}_X(2n)}{\bar{T}_X(n)}, \quad v_X(n) = \frac{\bar{V}_X(2n)}{\bar{V}_X(n)}$$

où $\bar{T}_X(n)$ (respectivement $\bar{V}_X(n)$) désigne la moyenne du temps de calcul (respectivement du volume de mémoire auxiliaire) utilisé pour le traitement par X d'une instance de taille n.

Attention, l'espace occupé par le tableau à trier ne doit pas être comptabilisé dans le calcul du volume de mémoire.

1. Déterminer $t_L(n)$, $t_R(n)$, $t_S(n)$ pour les algortihmes de tri : linéaire, rapide (quick-sort), et sélection.
2. Déterminer les fonctions $v_X(n)$ pour les mêmes algorithmes.
3. Quelles conclusions tirer de ces petits calculs ?

```
typedef struct {
  double real, img;
} cmp;

cmp prd( cmp x, cmp y)
{ cmp p;
  p.img = x.img  * y.real
        + x.real * x.img;
  p.real= x.real * y.real
        - x.img  * y.img;
  return p;
}
```

3. Optimisation à la Karatsuba.

1. Que calcule la fonction `prd` ?
2. S'inspirer de la méthode de Karatsuba pour utiliser une multiplication de moins.
3. On suppose que pour deux `double`, le temps de multiplication est λ-fois plus important que celui d'une addition ou d'une soustraction. Comparer les performances en fonction de λ.

4. Implantation du tri linéaire.

On suppose une implanation du trilinéaire en langage C

```
typedef unsigned long long ullong;
void trilin( ullong *t , ullong n );
```

```
[drm@obelix exam-12]$ lscpu
Architecture: i686    Mode: 32-bit Vitesse proc. en MHz: 3000.000

[drm@obelix exam-12]$ free -b
             total       used       free     shared
Mem:    1582931968  837398528  745533440          0
-/+ buffers/cache:  389427200 1193504768
Swap:   3187666944          0 3187666944

[drm@obelix exam-12]$ echo 'l(1582931968)/l(2)'  | bc -l
30.559952105962841097 56
```

FIG. 1 – capacités de la machine `obelix`

1. Quelles sont les préconditions d'utilisation ?
2. Estimer le temps de calcul d'une instance de taille 2^{20} sur la machine `obelix`.
3. pour la taille 2^{24} ?
4. pour la taille 2^{26} ?

5. Analyse d'une fonction récursive. On note $K(n)$, le nombre de comparaisons réalisées par la fonction récursive `proc` quand elle est appliquée à un tableau de n entiers.

1. Donner une formule récursive du temps de calcul.
2. Donner $K(n)$ en fonction de n.
3. Mézaufait, que fait cete fonction ?

```
int proc( int* t, int n )                    1
{  int x, y, p;                              2
   if ( n > 1 ) {                            3
      p = n / 2 + ( n & 1 );                 4
      x = proc( t,   n/2    );               5
      y = proc( t + n/2, p);                 6
      if ( x > y )  x = y;                   7
    }                                        8
   else x = *t;                              9
   return x;                                 10
}                                            11
```

6. Ordre d'une permutation. On considère les permutations de l'ensemble $\{0, 1, \ldots, n-1\}$. Une permutation π est représentée naturellement par un tableau de taille n. Pour un entier $x < n$, on définit la suite d'entiers $(x_n)_{n\geq 0}$

$$x_0 := x, \qquad \forall n > 0 \quad x_n = \pi(x_{n-1}),$$

on note alors $r(x)$ le plus petit entier strictement positif tel que $x_0 = x_{r(x)}$. L'ordre de π est le PPCM des valeurs de r, un tel ppcm peut être calculé de manière itérative :

$$\text{PPCM}(a_1, a_2, a_3, \ldots) = \text{PPCM}(\text{PPCM}(\text{PPCM}(a_1, a_2), a_3), \ldots)$$

1. Déterminer les valeurs de r, puis l'ordre de la permutation, correspondant au tableau de taille 15 :

   0 2 4 6 1 10 12 14 8 3 5 7 9 11 13

2. L'entier $r(x)$ est effectivement bien défini, pourquoi ?
3. Ecrire une fonction `ordre(p, n)` qui calcule l'ordre de la permutation p.