Preuve et Analyse des Algorithmes

10 janvier 2013

Q 1. Une implantation d'un algorithme traite une instance de taille 1024 en 1 seconde, une instance de taille 2048 en 4 secondes.

1. Quelle est la complexité probable de l'algorithme ?
2. Donner un exemple d'algorithme ayant cette complexité.

1. Quand la taille double, le temps de calcul est multiplié par 4, sans davantage d'informations, il est raisonnable de dire que la complexité est quadratique.
2. Le tri par sélection.

Q 2. L'algorithme [1] de tri du professeur Tee est un algorithme de tri comparatif qui trie un tableau de taille n en utilisant au plus $n(\log(\log(n)))^2$ comparaisons.

1. Une implantation de l'algorithme trie une instance de taille 65536 en 10^{-6} seconde. Estimer le temps de calcul d'une instance de taille 4294967296.
2. Que pensez du professeur Tee ?

1. On remarque que $65536 = 2^{16}$, $4294967296 = 2^{32}$

$$T(4294967296) = T(2^{32}) = 2^{32}5^2 = 2^{16}\frac{5^2}{2^2}T(65536) = 2^{12}5^2 10^{-6} = 2^{10}10^{-4} = 0.1024$$

2. Pour n grand

$$(\log\log n)^2 << \log n, \quad \lim_{x\to+\infty}\frac{\log x}{x} = 0.$$

La complexité de l'algorithme du professeur Tee contredit la limite théorique $\Omega(n\log n)$, il est forcément faux ! Mr Tee ferait mieux d'essayer le golf sur la wii !

Q 3. On considère l'algorithme de tri par sélection.

1. Donner une implantation en langage C.
2. Etablir une formule sur le nombre de comparaisons.

1.
```
void select( objet t[], int n )
{ int i, j, k;
  for( i = 0; i < n; i++ ){
      k = j;
      for( j = i+1; j < n; j++)
          if ( comp(t[j],t[k]) < 0 ) k = j;
      swap( t, i, k);
      }
}
```

[1] implantation disponible uniquement sous Macintosh (Mac OS 9 et Mac OS X avec iTunes 2.0)

2. On doit compter les comparaisons d'objets. L'algorithme passe par n étapes, l'étape i, génère $n-1-i$ comparaisons

$$C_{\text{sel}}(n) = \sum_{i=0}^{n-1}(n-i-1) = \sum_{i=0}^{n-1} i = \frac{n(n-1)}{2}$$

Q 4. Soit x un entier de n bits qui s'écrit en base 2 $(x_n \dots, x_2 x_1)$. Par définition le miroir sur n-bits de x est $\tilde{x} = (x_1 x_2 \dots, x_n)$.

1. Ecrire une fonction `mir(x,n)` en langage C pour caculer le miroir d'un entier $x < 2^n$
2. Préciser le domaine de validité de la fonction.
3. Préciser un contexte d'utilisation de cette notion.

1. Arithmétique binaire, pas de `pow`!!!

```
ullong mir( ullong x, int n )
{ ullong r = 0, bit = 1;
  bit <<= n-1;
  while ( x ) {
     if ( x & 1 ) r += bit;
     bit >>=1;
     x   >>=1;
  }
  return r;
}
```

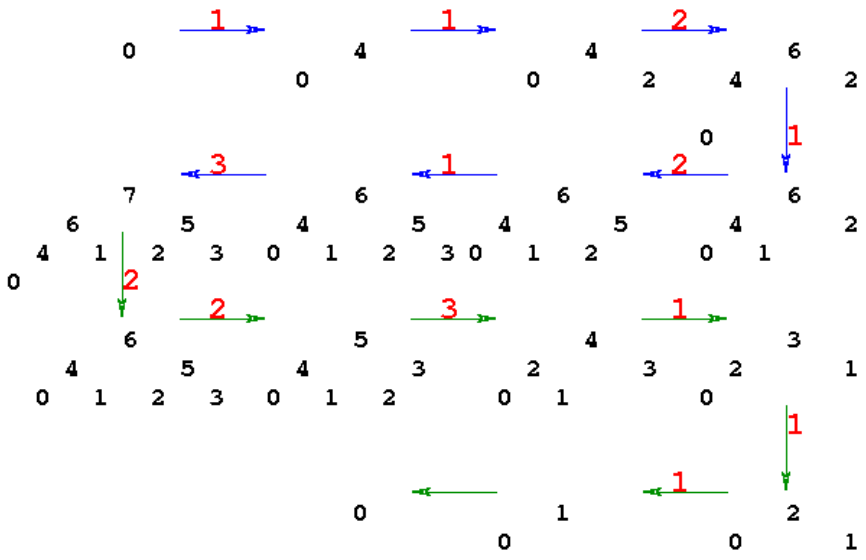
2. n doit être inférieur à 64.
3. Elle est au coeur de la FFT.

Q 5. On considère le tableau de 8 entiers

0 4 2 6 1 5 3 7

1. Détailler avec soin l'action du tri par tas sur ce tableau.
2. Combien de comparaisons ont été effectuées ?

1. Version naive du cours : 7 stalagmites, 7 stalactites !



2. Une vingtaine de comparaisons, c'est trés bien : $24 = 8 \log 8$.

Q 6. On considère les permutations de l'ensemble $\{0, 1, \ldots, n-1\}$. Une permutation π est représentée naturellement par un tableau de taille n. L'orbite de x suivant π est la plus petite partie de $\{0, 1, \ldots, n-1\}$ stable par π contenant x. Elle contient x, $\pi(x)$, puis $\pi(\pi(x))$ etc...

1. Déterminer les orbites correspondant au tableau de l'exercice précédent.
2. Ecrire une fonction `omega(x, p, n)` qui détermine la taille de l'orbite de x suivant la permutation p.
3. Ecrire une fonction `chi(p, n)` qui calcule le nombre d'orbites de la permutation p.
4. Préciser le temps de calcul.

1. Il y a 6 orbites :

$$\{0\}, \quad \{1,4\}, \quad \{2\}, \quad \{3,6\}, \quad \{5\}, \quad \{7\}.$$

2. ```
   int omega( int x, int p[], int n )
   { int r = 0, y;
     y= x;
     do {
        r++;
        y = p[x];
     } while ( x != y );
     return r;
   }
   ```

3. Pour économiser du temps de calcul, on mémorise les visites.

   ```
   int chi( int p[], int n )
   { int r = 0, x, y, *v;
     v = calloc( n, sizeof(int) );
     for( x = 0; x < n; x++ )
        if ( v[x] == 0 ) {
           r++;
           while( ! v[x] ) {
              v[x] = 1;
              x = p[x];
           }
        }
     free(v);
     return r;
   }
   ```

4. L'algorithme est linéaire en temps et en espace, analyse analogue au tri linéaire.
   ```