Algorithmique – écriture de programmes efficaces et sûrs

Romain Gille

08/02/2016

Rappel : Tri par sélection (TS)

Propriété sur laquelle TS est construit

Initialisation : k = 0

Condition d'arrêt : k = n (ou k = n-1)

Progression : `I(k)` et k $\neq$ n et a = `arg min T[k:n]` et `T[k]` = t_a et `t[a]` = t_k (permutation)

$\Rightarrow$ `I(k + 1)`

Notation :

`arg min T[0:k]` est la première valeur d'indice, a, telle que `T[a]` = min `T[0:k]`

De façon générale, `arg min` $f(x)$ est la plus petite valeur, a, telle que $f(a) = minf(x)$

On reprend le programme du cours précédent

```
void triSelection(int[] T){
  int n = T.length;
  int k = 0; // I(k)
  while(k != n){ // I(k) et k != n
    int m = indiceMin(T, k, n);
    int x = T[k];
    T[k] = T[m];
    T[m] = x; // I(k + 1)
    k = k + 1; // I(k)
  } // I(n) donc T[0:n] trié dans l'ordre croissant
}
```

Propriété de `I(a, k')` : a = `arg min T[k:k']`

Initialisation : a = k et k' = k + 1

Condition d'arrêt : k' = n

Progression :

`I(a, k')` et k' $\neq$ n et `T[k']` $\geq$ `T[a]` $\Rightarrow$ `I( a, k' + 1)`

`I(a, k')` et k' $\neq$ n et `T[k']` $<$ `T[a]` $\Rightarrow$ `I( k', k' + 1)`

```
int indiceMin(int[] T, int k, int n){ // retourne a = arg min T[k:n]
  int a = k, kp = k + 1; // I(a, k')
  while(kp != n){ // I(a, k') et k' != n
    if(T[kp] >= T[a]){ // I(a, k' + 1)
      kp ++; // I(a, k')
    }
    else{ // I(k', k' + 1)
      a = kp; // I(a, k' + 1)
      kp ++; // I(a, k')
    } // I(a, n) donc a = arg min T[k:n]
  }
}
```

Temps de calcul de l'appel `indiceMin(T, k, n)`

L'initialisation est en temps constant, α.

Le corps de boucle s'exécute en temps compté, β.

Le corps de boucle est exécuté `n - k - 1` fois le test `kp != n` est en temps constant, γ. Il évalué `n - k` fois.

D'où

$T_{indiceMin}(k) = \alpha + (n-k-1)\beta + (n-k)\gamma$

$= (\alpha - \beta) + (n-k)(\beta + \gamma)$

$= A + B(n-k)$

$= \Theta(n-k)$

```
void triSelection(int[] T){
  int n = T.length;                                    // temps constant : delta
  int k = 0; // I(k)                                   // temps constant : delta
  while(k != n){ // I(k) et k != n                     // A' + B(n - k)
    int m = indiceMin(T, k, n);                        // A' + B(n - k)
    int x = T[k];                                      // A' + B(n - k)
    T[k] = T[m];                                       // A' + B(n - k)
    T[m] = x; // I(k + 1)                              // A' + B(n - k)
    k = k + 1; // I(k)                                 // A' + B(n - k)
  } // I(n) donc T[0:n] trié dans l'ordre croissant    // A' + B(n - k)
}
```

$T_{TS}(n) = \delta + n(A' + \frac{B}{2}) + n^2 * \frac{B}{2}$

Tri par segmentation (CA Hoare)

Trier `T[i:j]`

1. "Installer" `T[i:k]` $\leq$ `T[k]` $<$ `T[K+1:n]`
2. Si `T[i:k]` est trié et `T[k + 1, j]` est trié. Alors `T[i:j]` est trié

```
void quickSort(int[] T){
  int m = T.length;
  qS(T, 0, n);
}

void qS(int[] T, int i, int j){ // si j-i <= 1 : T[i:j] est déjà trié
  if(j - i > 1){
    int k = segmenter(T, i, j); // T[i:k] <= T[k] < T[k+1:j]
    qS(T, i, k); // T[i:k] <= T[k] < T[k+1:j] et T[i:k] croissant
    qS(T, k + 1, j); // T[i:k] <= T[k] < T[k+1:j] et T[i:k] croissant et T[k+1:j] croissant
    // Donc T[i:j] croissant
  }
}
```

"DIVISER POUR REGNER"