Cours4
Algorithmes de tris – partie1

Année 2022-2023 J-F. Kamp - IUT de Vannes

PLAN

- Tri simple :
  - Principe
  - Algorithme
  - Efficacité de l’algorithme

- Tri rapide
  - Principe
  - Exemple
  - Algorithme de séparation
  - Algorithme de tri rapide

Tri simple

Principe

1000	129	-50	205	…	100
0	1	2	3	…	n-1

Trier par ordre croissant :

- étape1 : tableau de 0 à (n-1), placer la + petite valeur en « 0 »
- étape2 : tableau de 1 à (n-1), placer la + petite valeur en « 1 »
- étape3 : tableau de 2 à (n-1), placer la + petite valeur en « 2 »
- …
- étape(n-1) : tableau de (n-2) à (n-1), placer la + petite valeur en « (n-2) »

Algorithme

```
/* *
   *  Cette méthode trie un tableau d'entiers par ordre
   *  croissant des valeurs. La méthode est celle du tri
   *  simple (ramener les + petites valeurs en début de
   *  tableau).
   *  @param leTab le tableau des valeurs
   *  @param n le nombre de valeurs à trier
   */
void triSimple ( int [ ] leTab, int n ) {

   // variables locales
   int i, p, k;

   // initialisations
   i = 0;

   // première boucle = celle qui parcourt le tableau de 0
   // à (n – 2) y compris
   for ( i = 0; i <= (n-2); i++ ) {

      // sélectionner la + petite valeur sur le sous-tableau
      // allant de i à (n-1) : on identifie une case « k »

      // ensuite échanger cette case « k » avec « i »

   }

}
```

Algorithme

```
void triSimple ( int [ ] leTab, int n ) {

    // variables locales
    int i, p, k, min;

    // initialisations
    i = 0;

    // première boucle = celle qui parcourt le tableau de 0
    // à (n – 2) y compris
    for ( i = 0; i <= (n-2); i++ ) {

        // sélectionner la + petite valeur sur le sous-tableau
        // allant de i à (n-1) : on identifie une case « k »
        min = leTab[i];
        k = i;

        for ( p=(i+1); p <= (n-1); p++ ) {

            if ( leTab[p] < min ) {
                min = leTab[p];
                k = p;
            }

        }


        // ensuite échanger cette case « k » avec « i »

    }

}
```

Algorithme

```
void triSimple ( int [ ] leTab, int n ) {

    // variables locales
    int i, p, k, min, tmp;

    // initialisations
    i = 0;

    // première boucle = celle qui parcourt le tableau de 0
    // à (n – 2) y compris
    for ( i = 0; i <= (n-2); i++ ) {

        // sélectionner la + petite valeur sur le sous-tableau
        // allant de i à (n-1) : on identifie une case « k »

        …
        // k est l’indice de la case qui contient le min

        // ensuite échanger cette case « k » avec « i »
        tmp = leTab[k];

        leTab[k] = leTab[i];

        leTab[i] = tmp;

    }

}
```

Efficacité du tri simple

```
// Simplifions !
// boucle1
for ( i = 0; i <= (n-2); i++ ) {

    min = leTab[i];                       // 0 op. (faux !!)
    k = i;                                // 0 op. (faux !!)
    // boucle2
    for ( p=(i+1); p <= (n-1); p++ ) {

        // 1 seule op. (faux !!)
        if ( leTab[p] < min ) {
            min = leTab[p];
            k = p;
        }

    }

}
```

Coût de l'algorithme f(n) étant donné les simplifications :

- boucle1 :

  nb de tours : $n - 2 + 1 = n - 1$

- boucle2 :

  nb de tours : $n-1 - (i+1) + 1 = n - i - 1$

Efficacité du tri simple

Coût de l'algo. f(n) étant donné les simplifications :

- boucle1 :

  nb de tours : n – 2 + 1 = n – 1

- boucle2 :

  nb de tours : (n-1) - (i+1) + 1 = n – i – 1

$$f(n) \approx \sum_{i=0}^{n-2} (n - i - 1)$$

$$\approx (n-1) + (n-2) + \ldots + 1$$

Formule de math (à retenir !) :

$$1 + 2 + \ldots + m = m\,(m + 1) / 2$$

On pose m = (n-1) et f(n) devient :

$$f(n) \approx (\,n - 1\,)\,n / 2$$

Efficacité du tri simple

Coût de l'algorithme f(n) étant donné les simplifications :

$$f(n) \approx (n - 1) n / 2 = (n^2 - n) / 2$$

L'algorithme est donc en $\mathbf{\Theta(n^2)}$ pour n $\nearrow$

Pire des cas :
L'algorithme est trié « à l'envers » au départ.

```
if ( leTab[p] < min ) {     // 1 op.
    min = leTab[p];         // 1 op.
    k = p;                  // 1 op.
}
```

boucle2 :
nb de tours : (n – i – 1) x 3
$f(n) \approx 3 \times (n - 1) n / 2 = 3 \times (n^2 - n) / 2$

Tri rapide (QuickSort)

Principe du QuickSort

Diviser pour régner ou principe de séparation.

L’idée à chaque étape est de diviser le problème initial en sous-problèmes de même nature que le problème global, mais de taille + petite.

Chaque sous-problème est résolu par la même approche de séparation, et leurs solutions sont combinées pour donner la solution au problème global.

Principe du QuickSort

44	12	65	46	42	23	18	55
0	1	2	3	4	5	6	7

Problème global : trier ce tableau

Principe de séparation : si on parvient à placer une valeur correctement dans le tableau (cette valeur est appelée **pivot**), il suffit de ré-itérer ce placement sur 2 sous-tableaux :

- le tableau à droite du pivot
- le tableau à gauche du pivot

Et ainsi de suite jusqu'à n'avoir que des tableaux de taille 1 forcément triés.

Exemple pas à pas

44	12	65	46	42	23	18	55
0	1	2	3	4	5	6	7

Problème global : trier ce tableau

Pivot : 44 en case 0

Première séparation (à préciser…) :

18	12	23	42	**44**	46	65	55
0	1	2	3	**4**	5	6	7

Pivot = 44 est correctement placé

2 sous-tableaux :

- à gauche de 44 : de 0 à 3 (à trier)
- à droite de 44 : de 5 à 7 (à trier)

Exemple pas à pas

18	12	23	42
0	1	2	3

Sous-problème : trier ce tableau

Pivot : 18 en case 0

Séparation :

12	**18**	23	42
0	**1**	2	3

Pivot = 18 est correctement placé

2 sous-tableaux :

- à gauche de 18 : de 0 à 0 (tableau trié)
- à droite de 18 : de 2 à 3 (à trier)

Exemple pas à pas

23	42

2 3

Sous-problème : trier ce tableau

Pivot : 23 en case 2

Séparation :

23	42

2 3

Pivot = 23 est correctement placé

2 sous-tableaux :

- à gauche de 23 : tableau trié
- à droite de 23 : de 3 à 3 (tableau trié)

Exemple pas à pas

Ce n’est pas fini…

18	12	23	42	**44**	46	65	55
0	1	2	3	**4**	5	6	7

Partie à droite de 44

46	65	55
5	6	7

Exemple pas à pas

Partie à droite de 44

46	65	55
5	6	7

Sous-problème : trier ce tableau

Pivot : 46 en case 5

Séparation :

46	65	55
5	6	7

Pivot = 46 est correctement placé

2 sous-tableaux :

- à gauche de 46 : tableau trié
- à droite de 46 : de 6 à 7

Exemple pas à pas

65	55
6	7

Sous-problème : trier ce tableau

Pivot : 65 en case 6

Séparation :

55	**65**
6	**7**

Pivot = 65 est correctement placé

2 sous-tableaux :

- à gauche de 65 : de 6 à 6 (tableau trié)
- à droite de 65 : tableau trié

Exemple pas à pas

Au final, on recolle tous les tableaux à 1 case triés.

12	18	23	42	44	46	55	65
0	1	2	3	4	5	6	7

Le tableau est trié, le problème global est résolu.

Algorithme de séparation

44	12	65	46	42	23	18	55
0	1	2	3	4	5	6	7

Problème : placer le pivot au bon endroit dans le tableau tel que tout ce qui est à droite est >= pivot ET tout ce qui est à gauche est <= pivot.

Pivot : 44 en case 0

Départ

indL = 0 (case du pivot) (indice Left)

indR = (n-1) (dernière case) (indice Right)

Algorithme de séparation

Parcours de droite à gauche pour placer provisoirement le pivot.

droite –> gauche : avancer de indR vers indL tant que tab[indR] > pivot

ensuite échanger indL et indR

```
while ( tab[indR] > pivot ) {
        indR--;
}
echanger ( indR, indL );
```

18	12	65	46	42	23	**44**	55
0	1	2	3	4	5	**6**	7

Pivot « 44 » est provisoirement bien placé…

Il reste à examiner les cases de 1 à pivot.

Algorithme de séparation

Parcours de gauche à droite pour placer provisoirement le pivot.

indL = indL + 1 = 1

indR = 6

gauche –> droite : avancer de indL vers indR tant que tab[indL] < pivot

ensuite échanger indL et indR

```
while ( tab[indL] < pivot ) {
        indL++;
}
echanger ( indR, indL );
```

18	12	**44**	46	42	23	**65**	55
0	1	**2**	3	4	5	**6**	7

Pivot « 44 » est provisoirement bien placé…

Il reste à examiner les cases de 5 à pivot.

Algorithme de séparation

Parcours de droite à gauche pour placer provisoirement le pivot.

indR = indR **-** 1 = 5

indL = 2

droite –> gauche : avancer de indR vers indL tant que tab[indR] > pivot

ensuite échanger indL et indR

```
while ( tab[indR] > pivot ) {
        indR--;
}
echanger ( indR, indL );
```

18	12	**23**	46	42	**44**	65	55
0	1	**2**	3	4	**5**	6	7

Pivot « 44 » est provisoirement bien placé…

Il reste à examiner les cases de 3 à pivot.

Algorithme de séparation

Parcours de gauche à droite pour placer provisoirement le pivot.

indL = indL + 1 = 3

indR = 5

gauche –> droite : avancer de indL vers indR tant que tab[indL] < pivot

ensuite échanger indL et indR

```
while ( tab[indL] < pivot ) {
        indL++;
}
echanger ( indR, indL );
```

18	12	23	**44**	42	**46**	65	55
0	1	2	**3**	4	**5**	6	7

Pivot « 44 » est provisoirement bien placé…

Il reste à examiner les cases de 4 à pivot.

Algorithme de séparation

Parcours de droite à gauche pour placer provisoirement le pivot.

indR = indR - 1 = 4

indL = 3

droite –> gauche : avancer de indR vers indL tant que tab[indR] > pivot

ensuite échanger indL et indR

```
while ( tab[indR] > pivot ) {
        indR--;
}
echanger ( indR, indL );
```

18	12	23	**42**	**44**	46	65	55
0	1	2	**3**	**4**	5	6	7

Pivot « 44 » est provisoirement bien placé…

Il reste à examiner les cases de 4 à pivot.

Algorithme de séparation

18	12	23	**42**	**44**	46	65	55
0	1	2	**3**	**4**	5	6	7

Pivot « 44 » est DEFINITIVEMENT bien placé car indL (= 3 + 1 = 4) et indR (= 4) se sont rejoints.