

Support de cours

Module : Algo 1

Chapitre 7: Les algorithmes de tri et de recherche

Partie 2/2: Les algorithmes évolués

Réalisé par:
Dr. Sakka Rouis Taoufik

1

Chapitre 7 : Les algorithmes de tri et de recherche (Partie 2/2)

I. Le tri rapide

1) Principe

Le tri rapide (quick sort) est fondé sur la méthode de conception diviser pour régner en utilisant les deux étapes suivantes :

A- placer un élément du tableau (appelé pivot) à sa place définitive, en permutant tous les éléments de telle sorte que tous ceux qui sont inférieurs au pivot soient à sa gauche et que tous ceux qui sont supérieurs au pivot soient à sa droite. Cette opération s'appelle le partitionnement.

B- Pour chacun des sous-tableaux (sous tableau gauche et sous tableau droite), on définit un nouveau pivot et on répète l'opération de partitionnement. Ce processus est répété récursivement, jusqu'à ce que l'ensemble des éléments soit trié.

2

Chapitre 7 : Les algorithmes de tri et de recherche (Partie 2/2)

I. Le tri rapide

2) Réalisation

```
void main() {
    int n;
    float T[50];
    do {
        printf("Combien voulez-vous trier de données ? ");
        scanf("%d",&n);
    } while (n<=0 || n>50);

    remplirTab(t, n);
    tri_rapide(t,0,n-1);
    afficherTab(t, n);
}
```

3

Chapitre 7 : Les algorithmes de tri et de recherche (Partie 2/2)

I. Le tri rapide

2) Réalisation

```
void remplirTab (float t [ ], int n) {
    int i;
    printf("\n Entrez les %d données \n ", n);

    for (i=0; i<n; i++)
        scanf("%f", &t[i]);
}
```

4

Chapitre 7 : Les algorithmes de tri et de recherche (Partie 2/2)

I. Le tri rapide

2) Réalisation

```
int partition (float t [ ], int gauche, int droite) {
    float cle;
    int i , j ;
    cle= t [gauche];
    i=gauche+1;
    j=droite;
    while (i<=j)    {
        while (t[i]<=cle) i++;
        while (t[j]>cle) j--;
        if (i<j) echanger(t, i++, j--);
    }
    echanger(t, gauche, j);
    return j;
}
```

5

Chapitre 7 : Les algorithmes de tri et de recherche (Partie 2/2)

I. Le tri rapide

2) Réalisation

```
void echanger (float t [ ], int i, int j) {
    float aux;
    aux=t[i];
    t [i]=t[j];
    t [j]= aux;
}
void afficherTab(float t [ ], int n) {
    int i;
    printf("\n Voici le tableau trié \n");
    for (i=0; i<n; i++)
        printf("%6.2f ",t [i]);
    printf("\n");
}
```

6

Chapitre 7 : Les algorithmes de tri et de recherche (Partie 2/2)

I. Le tri rapide

2) Réalisation

```
void tri_rapide(float t [ ] , int gauche, int droite) {
    int pivot;

    if (gauche<droite)
    {
        pivot=partition(t, gauche, droite);
        tri_rapide(t, gauche, pivot-1);
        tri_rapide(t, pivot+1, droite);
    }
}
```

7

Chapitre 7 : Les algorithmes de tri et de recherche (Partie 2/2)

I. Le tri rapide

3) Complexité

Rappelant qu'il existe plusieurs algorithmes permettant de réaliser l'opération de tri.

Ces algorithmes sont classés en deux catégories :

Algorithmes élémentaires : tri par sélection, tri par insertion, etc. (complexité n^2)

Algorithmes évolués : tri par tas, tri rapide, etc.

(complexité $n \log_2(n)$)

(démonstration par récurrence)

8

Chapitre 7 : Les algorithmes de tri et de recherche (Partie 2/2)

II. La recherche Dichotomique

1) Présentation informelle

La table est triée par ordre croissant sur une caractéristique. L'algorithme de recherche séquentielle est sur une table triée. Mais on peut faire mieux, en proposant l'algorithme suivant :

On compare « x » par rapport au milieu de notre espace de recherche, si sa coïncidence avec cet élément, alors on s'arrête avec un résultat positif sinon on s'intéresse soit au sous tableau de gauche (si $x <$ à l'élément du milieu), soit au sous tableau de droite (si $x >$ à l'élément du milieu). Et on respecte ces opérations jusqu'à le sous tableau de recherche soit vide.

9

Chapitre 7 : Les algorithmes de tri et de recherche (Partie 2/2)

II. La recherche Dichotomique

2) Réalisation en C

```
#include <string.h>
```

```
int recherche_dichotomique(float T [ ] , int n, float x) {
```

```
/*cette fonction retourne -1 si x n'appartient pas au tableau,
retourne la position m si T[m]=x */
```

```
int g, d ;/*sous tableau de recherche*/
```

```
int m ;/*indice de l'élément au milieu*/
```

```
/*initialisation*/
```

```
g=0 ;
```

```
d=n-1 ;
```

10

Chapitre 7 : Les algorithmes de tri et de recherche (Partie 2/2)

II. La recherche Dichotomique

2) Réalisation en C

```

do {   m=(g+d)/2 ;/*division entière*/
      if(T[m]==x)
          return m ; /*issue positive*/
      /*soit déplacer g soit déplacer d jamais déplacer les
      deux à la fois*/
      if(T[m]>x)
          d=m-1;
      else
          g=m+1 ;
}while(g<=d) ;
/*g>d sous tableau vide*/
return -1 ; /*issue négative*/
}

```

11

Chapitre 7 : Les algorithmes de tri et de recherche (Partie 2/2)

II. La recherche Dichotomique

3) Complexité

-Complexité en temps :

On va comptabiliser l'opération de comparaison de x avec T[m]) ?

On distingue les cas suivant :

-Cas minimum ou optimiste: une seul comparaison, ceci traduit que x coïncide avec T[0+(n-1)/2].

-Cas maximum ou pessimiste : (dans le pire des cas) un tel cas traduit x n'appartient pas à T. À chaque itération, on part d'un problème de taille N et moyennant une comparaison, on tombe sur un problème de taille N/2. L'algorithme de recherche dichotomique applique le principe « diviser pour résoudre » ou encore « diviser pour régner » : le problème initiale est divisé en deux sous problème **disjoints** et de taille plus ou moins égale.

12

Chapitre 7 : Les algorithmes de tri et de recherche (Partie 2/2)

II. La recherche Dichotomique

3) Complexité

On note C_N : le nombre de fois où `strcmp` est effectuée.

$$C_n = C_{n/2} + 1$$

avec $C_1 = 1$

On pose $N=2^n$ ou $n=\log_2 N$

$$C_2^n = C_2^{n-1} + 1$$

$$= C_2^{n-2} + 1 + 1$$

$$= C_2^{n-3} + 3$$

.....

$$= C_1 + n = 1 + n = 1 + \log_2 N$$

Ainsi, cet algorithme est $O(\log_2 N)$

-Cas moyen : entre 1 et $\log_2 N$

13

Chapitre 7 : Les algorithmes de tri et de recherche (Partie 2/2)

II. La recherche Dichotomique

3) Complexité

Remarque : le gain apporté par l'application de l'algorithme de recherche dichotomique sur un tableau **trié** peut être illustré par l'exemple suivant :

On souhaite effectuer une recherche sur un tableau trié T de taille $N=10000$.

Si on applique l'algorithme de recherche séquentielle la complexité dans le cas moyen est 5000

l'algorithme de recherche dichotomique, la complexité au pire des cas est $O(\log_2 10000) \approx 14$

14

 Chapitre 7 : Les algorithmes de tri et de recherche (Partie 2/2)

III. Exercices d'application

Proposer une fonction récursive basée sur l'algorithme de recherche dichotomique qui permettra de localiser la position d'un entier x dans un tableau trié d'entiers. La recherche dichotomique est une méthode de recherche efficace qui divise le problème en deux à chaque étape, en éliminant la moitié des éléments non pertinents à chaque itération.

15