Cours de Programmation Impérative
Recherches et tris

Julien David

A101 - david@lipn.univ-paris13.fr

Recherche dans un tableau

Ne supposons rien . . .

. . . ou presque

- On suppose que le tableau contient uniquement des entiers.
- On suppose que les valeurs apparaissent dans n'importe quel ordre.

On veut une fonction qui

- prend en entrée : un entier x, un tableau t, une taille n,
- renvoie 1 si x appartient au tableau,
- renvoie 0 si x n'appartient pas au tableau.

Dans ce cas

- Au pire, on est obligé de vérifier toutes les cases du tableau.
- La complexité en temps est donc $\mathcal{O}(n)$.

Une fonction itérative

```
1 int recherche_element(int * tab,int n,int x){
2   int i=0;
3   while(i<n){
4     if(tab[i]==x)
5       return i;
6     i++;
7   }
8   return -1;
9 }
```

Vu au cours précédent

- Complexité en temps : $\mathcal{O}(n)$ et $\Omega(1)$.
- Complexité en espace : $\Theta(1)$.

Supposons maitenant . . .

. . . que le tableau est trié

- On suppose que le tableau est trié dans l'ordre croissant.

Dans ce cas

- Il n'y a plus obligation de vérifier toutes les cases du tableau.

Une première fonction

```
1 int recherche_element_trie(int * tab,int n,int x){
2   int i=0;
3   while(i<n&&tab[i]<=x){
4     if(tab[i]==x)
5       return i;
6     i++;
7   }
8   return -1;
9 }
```

Ce qui ne change pas grand chose . . .

- Complexité en temps : $\mathcal{O}(n)$ et $\Omega(1)$.
- Complexité en espace : $\Theta(1)$.

Utilisons cette information

Dans un tableau trié

Je cherche à savoir si un élément x appartient à un tableau **trié** de n cases.

-10	-4	3	5	6	8	23	112

- si $x < tab\left[\frac{n}{2}\right]$, alors x ne peut pas être dans la partie droite du tableau.

  Exemple : $x = 0$

-10	-4	3	5	6	8	23	112

Utilisons cette information

Dans un tableau trié

Je cherche à savoir si un élément x appartient à tableau de n cases.

-10	-4	3	5	6	8	23	112

- si $x > tab\left[\frac{n}{2}\right]$, alors x ne peut pas être dans la partie gauche du tableau.

  Exemple : $x = 47$

-10	-4	3	5	6	8	23	112

En conséquence

- Il est possible, en **un seul test** de diviser l'espace de recherche par 2 !

Fonction récursive

```
1  int recherche_rec(int * tab,int n,int x){
2    int milieu=n/2;
3    if(n>0){
4      if(x>tab[milieu]){
5        if(n%2==1)
6          return recherche_rec(&tab[milieu+1],milieu,x);
7        else
8          return recherche_rec(&tab[milieu+1],milieu-1,x);
9      }
0      else if(x<tab[milieu])
1          return recherche_rec(tab,milieu,x);
2      return 1;
3    }
4    return 0;
5  }
```

Fonction récursive : complexité

Complexité de la fonction

- Complexité en temps : $\mathcal{O}(\log n)$ et $\Omega(1)$
- Complexité en espace : $\mathcal{O}(\log n)$ et $\Omega(1)$

Fonction itérative

```
1  int recherche_it(int * tab,int n,int x){
2    int debut=0;
3    int taille=n;
4    int milieu;
5    while(taille>0){
6      milieu=debut+taille/2;
7      if(tab[milieu]==x)
8        return 1;
9      if(x>tab[milieu]){
0        debut=milieu+1;
1        if(taille%2==1)
2          taille=taille/2;
3        else
4          taille=taille/2-1;
5      }
6      else
7        taille=taille/2;
8    }
9    return 0;
20 }
```

Fonction itérative : complexité

Complexité de la fonction

- Complexité en temps : $\mathcal{O}(\log n)$ et $\Omega(1)$
- Complexité en espace : $\Theta(1)$

Moralité

Pour avoir des programmes efficaces . . .

- Il est souvent préférable d'avoir des données triées, ou "ordonnées".

. . . ça tombe bien !

On travaille sur des ordinateurs.

- Ordinateur : du latin ordinator ("celui qui met de l'ordre").

Crier le tableau

Les algorithmes

Il existe une multitude d'algorithmes de tri :

- le tri par sélection
- le tri à bulles
- le tri fusion
- le tri rapide (*quicksort*)
- ...

Le tri par sélection

L'idée

On part d'un tableau non trié de taille n.

1. on cherche le plus petit élément du tableau (*min*),
2. on échange la première case du tableau et la case contenant *min*,
3. on recommence avec le tableau de taille $n - 1$.

Le tri par sélection : code

```
1  int recherche_position_min(int *tab, int n){
2    int i;
3    int pos_min=0;
4    for(i=1;i<n;i++){
5      if(tab[i]<tab[pos_min])
6        pos_min=i;
7    }
8    return pos_min;
9  }
0
1  void tri_selection(int * tab,int n){
2    int i;
3    int pos_min;
4    for(i=0;i<n;i++){
5      pos_min=recherche_pos_min(&(tab[i]),n-i);
6      swap(&tab[i],&tab[pos_min]);
7    }
8  }
```

Le tri par sélection : complexité

Recherche du minimum

- Complexité en temps : $\Theta(n)$.
- Complexité en espace : $\Theta(1)$.

Tri par seléction

- On effectue n fois la recherche d'un minimum.
- Complexité en temps : $\Theta(n^2)$.
- Complexité en espace : $\Theta(1)$.

Le tri à bulles : l'idée

L'idée

- On parcours le tableau entièrement.
- Dès qu'on rencontre une case où la valeur est plus grande que celle de sa voisine de droite, on échange les valeurs.
- Si lors du parcours on a echangé au moins deux valeurs, on recommence un nouveau parcours.

Le tri à bulles : code

```
1  #define VRAI 1
2  #define FAUX 0
3  void tri_bulle(int * tab,int n){
4    int echange;
5    int i;
6    do{
7      echange=FAUX;
8      for(i=0;i<(n-1);i++){
9        if(tab[i]>tab[i+1]){
0          swap(&tab[i],&tab[i+1]);
1          echange=VRAI;
2        }
3      }
4    }while(echange==VRAI);
5  }
```

Le tri à bulles : complexité

Tri à bulles

- Complexité en temps : $\mathcal{O}(n^2)$ et $\Omega(n)$
- Complexité en espace : $\Theta(1)$

Idée de la preuve

- La boucle `for` effectue $\Theta(n)$ intructions.
- La boucle `while` :
  - Au mieux : le tableau est déjà trié et aucun échange n'est fait.
  - Au pire : à la fin de la i-ème itération du while, on sait que les i derniers éléments du tableaux sont à leur place. Il y a donc au plus n itérations.

Tableaux de truc

Recherche et tri de trucs

Facile à adapter

Pour adapter les fonctions vues aujourd'hui à n'importe quel type de données, il suffit de pouvoir :

- Comparer deux éléments de même type ($<, ==, >$),
- Échanger les valeurs de deux éléments.

Alors trions des trucs

- En TD, on inventera pleins de trucs.
- Pour chaque truc on fera une fonction `cmp_truc` et `swap_truc`.
- Et on pourra trier des trucs.