Algorithmique

Parcours et tri d'un tableau

Yannick CHISTEL

Lycée Dumont d'Urville - CAEN

Mars 2020

Algorithme

Définition

Un **algorithme** est l'écriture en langage naturel d'une suite d'instructions répondant à une problématique donnée.
Un algorithme peut être traduit dans n'importe quel langage de programmation tout en respectant la structure globale de celui-ci.
Des mots clés seront utilisés comme :

- **si**, **alors** et **sinon** pour traiter les structures conditionnelles ;
- **Pour** et **Tant que** pour traiter les structures itératives ;
- L'**affectation** d'une valeur à une variable se note par la flèche $\longleftarrow$

Exemple

Voici un algorithme de recherche de la valeur minimale dans un tableau T :

$m \longleftarrow T[0]$
Pour i allant de 1 à longueur(T) :
    Si $T[i] < m$ alors $m = T[i]$
Fin pour

Algorithme de tri

Introduction

Il est souvent plus simple de traiter des données lorsque celles-ci sont triées. Il existe deux algorithmes pour trier des valeurs : le **tri par sélection** et le **tri par insertion**.

Principe du tri par sélection

On sépare en 2 parties les données à trier. La première partie contient les données déjà triées et la seconde contient les données non triées.

On cherche la plus petite valeur dans la seconde partie et on la place au début de celle-ci. Elle devient alors la plus grande valeur de la première partie triée.

On recommence jusqu'à avoir la seconde partie vide.

Principe du tri par insertion

On sépare en 2 parties les données à trier. La première partie contient les données déjà triées et la seconde contient les données non triées.

On prend la première valeur de la partie non triée (la plus à gauche) et on l'insère dans la partie triée en décalant d'un rang vers la droite toutes les valeurs qui lui sont supérieures.

On recommence jusqu'à avoir la seconde partie vide.

Algorithme de tri par sélection

Algorithme

Voici une écriture de l'algorithme de tri par sélection :

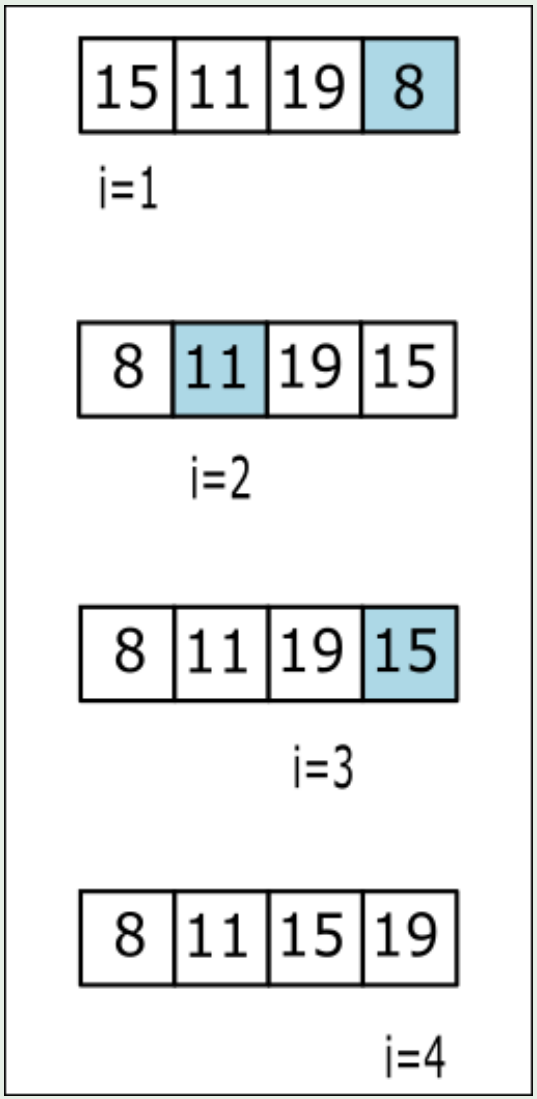
```
i ⟵ 1
tant que i < longueur(t) : //boucle 1
    j ⟵ i
    min ⟵ i
    tant que j <= longueur(t) : //boucle 2
        si t[j] < t[min] :
            min ⟵ j
        fin si
    j ⟵ j + 1
    fin tant que
    si min ≠ i :
        échanger t[i] et t[min]
    fin si
    i ⟵ i + 1
fin tant que
```

Algorithme de tri par sélection

Exemple

Soit le tableau de valeurs $T = [15, 11, 19, 8]$

1. $i = 1$: on cherche la valeur minimale du tableau, entre $j = 1$ et $j = 4$: ici c'est 8 pour $j = 4$. On mémorise l'indice dans la variable min $= 4$.
   $i \neq$ min, on échange les 2 valeurs : $8 \longleftrightarrow 15$
2. $i = 2$, on cherche la valeur minimale du tableau, entre $j = 2$ et $j = 4$: ici c'est pour $j = 2$. On mémorise l'indice dans la variable min $= 2$.
   $i =$ min $= 4$, on n'échange pas les valeurs.
3. $i = 3$, on cherche la valeur minimale du tableau, entre $j = 3$ et $j = 4$: ici c'est pour $j = 4$. On mémorise l'indice dans la variable min $= 4$.
   $i \neq$ min, on échange les 2 valeurs : $19 \longleftrightarrow 15$
4. $i = 4$, on cherche la valeur minimale du tableau, entre $j = 4$ et $j = 4$ donc pour $j = 4$. On mémorise l'indice dans la variable min $= 4$.
   $i =$ min $= 4$, on n'échange pas les valeurs.



Algorithme de tri par insertion

Algorithme

Voici une écriture de l'algorithme de tri par insertion :$j \longleftarrow 2$

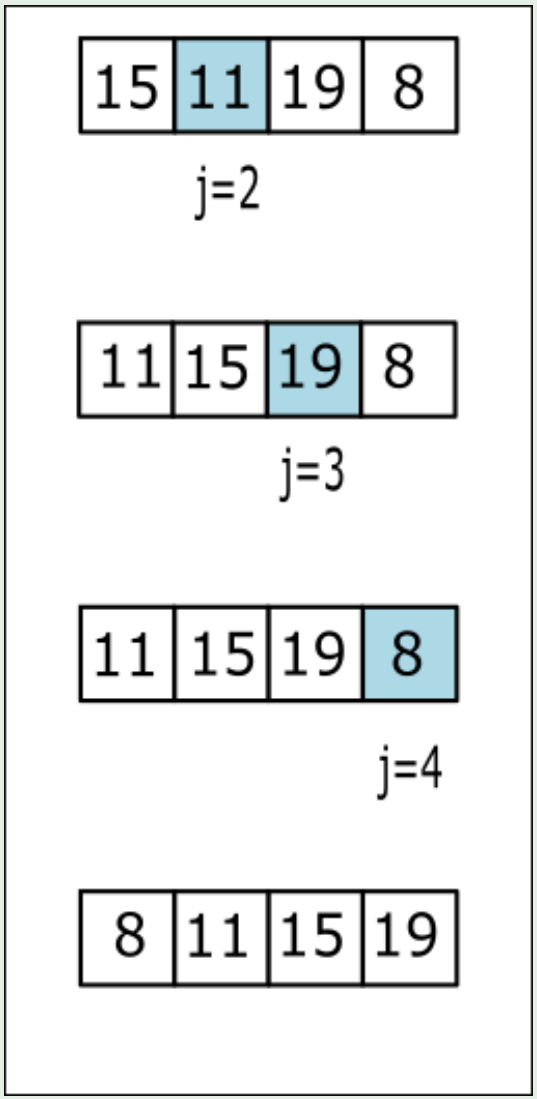
tant que $j <= \text{longueur}(t)$: //boucle 1
    $i \longleftarrow j - 1$
    $k \longleftarrow t[j]$
    tant que $i > 0$ et que $t[i] > k$: //boucle 2
        $t[i+1] \longleftarrow t[i]$
        $i \longleftarrow i - 1$
    fin tant que
    $t[i+1] \longleftarrow k$
    $j \longleftarrow j + 1$
fin tant que

Algorithme de tri par insertion

Exemple

Soit le tableau de valeurs $T = [15, 11, 19, 8]$

1. $j = 2$, $k = T[2] = 11$ et $i = 1$: on compare la valeur 11 avec la valeur 15. Comme elle est inférieure on décale à droite la valeur 15 en $j = 2$ et on insère la valeur 11 en $i = 1$.
2. $j = 3$, $k = T[3] = 19$ et $i = 2$: on compare la valeur 19 avec la valeur 15. Comme elle est supérieure on sort de la boucle ($T[2] > k$), on garde $T[3] = 19$ et on passe à $j = 4$.
3. $j = 4$, $k = T[4] = 8$ et $i = 3$:
   - on compare la valeur 8 avec la valeur 19. Comme elle est inférieure on décale à droite la valeur 19 en $i = 4$ et i passe à 2 ;
   - on compare la valeur 8 avec la valeur 15. Comme elle est inférieure on décale à droite la valeur 15 en $i = 3$ et i passe à 1 ;
   - on compare la valeur 8 avec la valeur 11. Comme elle est inférieure on décale à droite la valeur 11 en $i = 2$ et i passe à 0. On sort de la boucle et $T[1] = 8$.



Introduction

Les langages de programmation possèdent souvent leurs propres fonctions de tri.
En python, il existe pour les tableaux, 2 fonctions de tri qui sont **sort()** et **sorted()**

1. La fonction **sort()** s'applique sur un tableau et trie ses valeurs par ordre croissant.
   Le tableau initial est modifié.

```
A=[3,1,5,2]
A.sort()
A
```

```
[1, 2, 3, 5]
```

2. La fonction **sorted()** prend en argument un tableau et crée un nouveau tableau avec les valeurs du tableau initial triées dans l'ordre croissant.
   Le tableau initial n'est pas modifié.

```
B=[3,1,5,2]
C=sorted(B)
C,B
```

```
([1, 2, 3, 5], [3, 1, 5, 2])
```

Recherche dichotomique

Principe

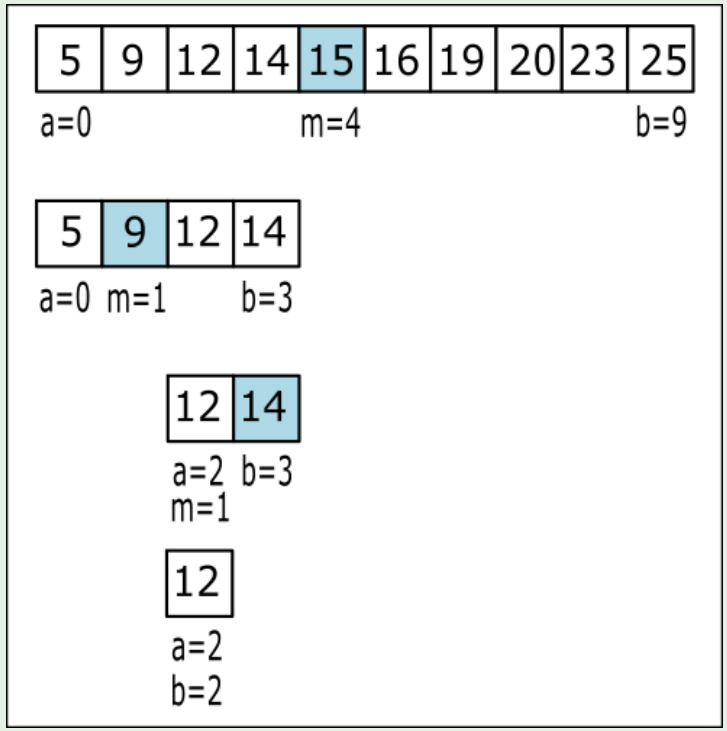
La **dichotomie** est un mot d'origine greque qui signifie ≪ diviser en deux ≫.
La recherche d'une valeur dans un tableau peut être facilitée si celui-ci est trié. Dans ce cas, on divise successivement le tableau en 2 jusqu'à atteindre la valeur cherchée.

Exemple

Soit $T = [5, 9, 12, 14, 15, 16, 19, 20, 23, 25]$

1. On cherche le nombre 12. Il est inférieur à la plus grande valeur du tableau.
2. On calcule la valeur de l'indice situé au milieu du tableau : $m = 4$.

   On compare la valeur $T[4]$ avec 12 :
   - si $12 < T[4]$ alors 12 est dans la première moitié du tableau ;
   - si $12 > T[4]$ alors 12 est dans la seconde moitié du tableau ;
   - si $12 = T[4]$ alors le nombre est trouvé
3. On répète ce processus jusqu'à avoir un tableau réduit à une valeur ou vide !



Recherche dichotomique

Algorithme

Voici une écriture de l'algorithme de recherche dichotomique dans un **tableau trié** :

t désigne un tableau trié
v est la valeur cherchée dans le tableau
a, b et m sont les indices de position des valeurs dans le tableau.

$a \longleftarrow 1$ *...première valeur du tableau*
$b \longleftarrow$ longueur(t) *...dernière valeur du tableau*
tant que $a <= b$:
    $m \longleftarrow (a+b)//2$ *...m est la position au milieu*
    si $v < t[m]$ alors $b = m-1$ *...v se trouve dans la première moitié*
    sinon si $v > t[m]$ alors $a = m+1$ *...v se trouve dans la seconde moitié*
    sinon la valeur est trouvée en m
fin tant que
La valeur n'est pas trouvée

Terminaison de l'algorithme

Variant de boucle

On appelle **variant de boucle** une quantité entière qui :

- doit être positive ou nulle pour rester dans la boucle ;
- décroit strictement à chaque itération

Si on trouve une telle quantité dans une boucle while, celle-ci se termine.

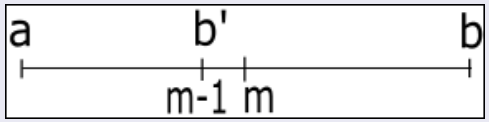
Preuve de la terminaison

Dans l'algorithme de recherche par dichotomie, le variant de boucle est $b - a$.

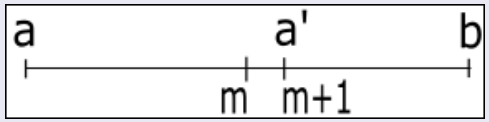
- Ce nombre est clairement supérieur ou égal à 0 puisque $a <= b$.
- Vérifions qu'elle décroît en distinguant 3 cas :

cas 1 : $t[m] == v$ alors on sort de la boucle.

cas 2 : $t[m] > v$ donc $b' - a < m - a < b - a$ donc la quantité $b - a$ décroit.



cas 3 : $t[m] < v$ donc $b - a' < b - m < b - a$ donc la quantité $b - a$ décroit.



$b - a$ est un variant de boucle positif qui décroit, assurant la terminaison de la **boucle while**.

Coût, efficacité, complexité d'un algorithme

Introduction

Déterminer l'efficacité d'un algorithme est important. Certaines instructions sons répétées de nombreuses fois et peuvent finir par prendre beaucoup de temps. On cherche alors à calculer un ordre de grandeur du nombre de calculs réalisés.

1. La recherche d'une valeur dans un tableau (minimum, maximum) est de complexité **linéaire**. Le nombre de calculs (comparaisons) est proportionnel à la dimension n du tableau. On note cette complexité par $O(n)$.
2. Le tri d'un tableau par sélection ou insertion est de complexité **quadratique**. Le nombre d'opérations est proportionnel au carré de la dimension n du tableau. On note cette complexité par $O(n^2)$.
3. La recherche par dichotomie est de complexité logarithmique. On la note $O(\log_2(n))$.

Propriété

On peut comparer l'efficacité des algorithmes en comparant leur complexité. Pour un tableau de dimension n, on a :

$$O(\log_2(n)) < O(n) < O(n^2)$$