# Algorithmie - parcours séquentiel

## Qu'est-ce que l'algorithmie

L'algorithmie c'est l'étude des algorithmes. C'est-à-dire des façons de résoudre certains problèmes et d'étudier leur coût. Nous ne nous intéresserons qu'au coût que l'on appelle la complexité en temps, c'est-à-dire le nombre d'opérations élémentaires à faire en fonction de la taille des données. Chaque opération élémentaires prenant un certain temps on peut estimer le temps que va mettre un algorithme à résoudre un problème en fonction de la taille des données.

En algorithmie, on ne s'intéresse pas à l'implémentation d'un algorithme. Nous avons vu qu'un tableau peut se parcourir par indice ou par élément. C'est une différence d'implémentation donc nous nous en occuperons pas bien que cela puisse avoir un certain impact sur la vitesse exécution.

## Parcours séquentiel d'un tableau

Le parcours séquentiel d'un tableau est une famille d'algorithmes qui permet de résoudre des problèmes comme la recherche d'extrémum, la recherche d'un élément, le nombre d'occurrences d'un élément, la somme ou la moyenne de nombre, etc.

Parcourir un tableau séquentiellement, c'est lire ses éléments un par un en partant du premier élément. Il y a deux cas de figure, soit il faut lire tous les éléments du tableau comme par exemple dans la recherche d'un maximum, soit partiellement comme dans la recherche d'un élément ou l'on peut s'arrêter dès que l'élément est trouvé. Vous trouverez en dessous un certain nombre d'algorithmes qui utilisent un parcours séquentiel d'un tableau.

## I Un premier exemple

La fonction `recherche` prend en paramètre un tableau `tab` de nombres et un nombre `elem`. Elle retourne `True` si `elem` appartient au tableau `tab` et `False` sinon.

### Principe

Le tableau est parcouru élément par élément. Si l'élément recherché est trouvé `True` est retourné.

Si tous les éléments ont été parcouru et qu'aucun n'est égal à l'élément recherché `elem` alors `False` est retourné.

Le tableau n'est donc parcouru en entier que lorsque l'élément recherché n'est pas contenu dans le tableau. Sinon il parcourt le tableau jusqu'au premier élément recherché trouvé.

### En pseudo code

```
fonction recherche (tab, elem):
    pour chaque élément e du tableau tab:
        si e est égal à elem:
            retourner Vrai
    retourner Faux
```

### En Python

```python
def recherche(tab,elem):
    for e in tab:
        if e == elem:
            return True
    return False

# 5 appartient-il au tableau [1,2,3,4]?
print(recherche([1,2,3,4],5))
```

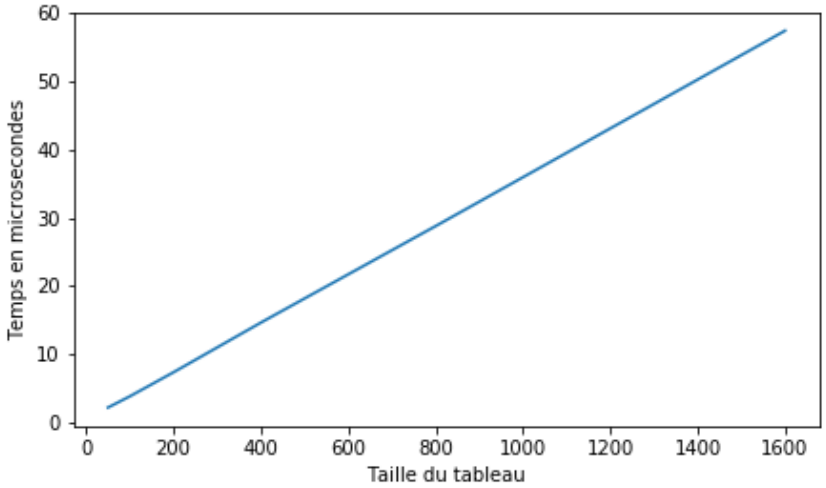
Voir dans PythonTutor

Il est aussi possible de programmer cette fonction par indice:

```python
def recherche(tab,elem):
    for i in range(len(tab)):
        if tab[i] == elem:
            return True
    return False
```

Si l'on mesure le temps d'exécution en fonction de la taille du tableau dans le pire des cas, c'est-à-dire lorsque l'élément n'est pas dans le tableau et que l'on doit parcourir le tableau en entier.

On obtient ce graphique:




**À retenir:**

Le parcours séquentiel d'un tableau a un **coût linéaire**, c'est-à-dire que le temps d'exécution est proportionnel à la taille du tableau parcouru. On note $\mathrm{O}(n)$ se lit "grand O de $n$"

Dans un tableau *non trié* c'est souvent la seule façon de trouver un élément, le plus grand, la somme de tout les éléments ou encore par exemple le nombre de fois ou un certain élément est rencontré.

## II Algorithmes de comptage

La fonction `nb_occurence` retourne le nombre d'éléments égaux à `elem` dans le tableau `tab`.

### Principe

On utilise un accumulateur qui va compter le nombre de fois où l'élément `elem` rencontré.

L'accumulateur `nb_occ` est initialisé à 0.

Le tableau est parcouru élément `e` par élément. À chaque fois que qu'un élément égal à `elem` est rencontré le compteur `nb_oc` est incrémenté de 1. Une fois chacun des éléments parcouru, l'accumulateur `nb_occ` est retourné.

#### En pseudo code

```
fonction nb_occurence(tab, elem):
    nb_occ = 0  # L'accumulateur est initialisé à 0
    pour chaque élement e du tableau tab:
        si e est égal  à elem:
            on incrémente de 1 nb_occ
    retourner nb_occ
```

#### En Python

```
def nb_occurence(tab,elem):
    nb_oc = 0
    for e in tab:
        if e == elem:
            nb_oc = nb_oc +1
    return nb_oc
print(nb_occurence([1,2,3,1,1,5],1))
```

Ce programme affiche `3` . Il y a bien trois 1 dans le tableau.

## III Algorithmes de cumul

### Exemple de calcul moyenne

La fonction `moyenne` retourne la moyenne des éléments du tableau `tab` donné en argument.

#### Principe

On utilise un accumulateur `somme` qui va contenir la somme des éléments contenus dans le tableau `tab` .

L'accumulateur `somme` est initialisé à 0.

Pour chaque élément `note` du tableau `tab` , on ajoute à `somme` la valeur de `note`.

Un fois tous les éléments parcourus, la fonction retourne le quotient de la somme des notes contenu dans `somme` par `len(tab)` qui est la longueur du tableau et égal au nombre de notes.

#### En pseudocode

```
fonction moyenne(tab):
    somme  = 0 # somme est initialisée à 0
    Pour chaque note du tableau:
        On ajoute à somme la valeur de note.
    Retourner somme / longueur de tab
```

### En Python

```
def moyenne(tab):
    somme = 0
    for note in tab:
        somme = somme + note
    return somme / len(tab)

print(moyenne([8,12,10,14]))
```

Ce programme affiche `11.0` qui est la moyenne de 8, 12, 10 et 14.

Voir dans PythonTutor

### Exemple de comptage

La fonction `nb_mots_avec_a` pour paramètre un tableau `tab` et retourne le nombre de mots contenant la lettre `'a'` dans ce tableau.

### Principe

L'accumulateur `nb_mots` qui est ici un compteur est initialisé à 0.

Puis pour chaque mot du tableau, s'il contient la lettre `'a'` l'accumulateur `nb_motsest` incrémenté de 1.

Un fois tout le tableau parcouru `nb_mots` est retourné.

### En pseudocode

```
fonction  nb_mots_avec_a(tab):
    nb_mots  = 0 # Le compteur est initialisé à 0
    pour chaque mot de tab:
        si mot contient la lettre  'a':
            nb_mots = nb_mots + 1
    retourner  nb_mots
```

### En python

```
def nb_mots_avec_a(tab):
    nb_mots = 0
    for mot in tab:
        if 'a' in mot:
            nb_mots = nb_mots + 1
    return nb_mots

print(nb_most_avec_a(["été","salé","sucré","lait","parler"]))
```

Ce programme affiche `3` . Il y a bien trois des cinq mots qui contiennent la lettre `'a'` .

Voir dans PythonTutor

## IV Algorithme de recherche de maximum ou de minimum

La fonction `maximum` a pour paramètre un tableau et retourne le plus grand élément de celui-ci.

### Principe

La variable `maxi` est initialisé à la valeur du premier élément.

Les éléments sont ensuite parcourus un par un. Dès qu'un élément est plus grand que `maxi`, cette valeur est affectée à `maxi`

. Une fois tout les éléments parcouru la fonction retourne `maxi`

#### En pseudocode

```
fonction maximum(tab):
    maxi est initialisé avec la première valeur du tableau tab.
    Pöur chaque élement e du tableau tab:
        Si e >  maxi:
            maxi = e
    Retourner maxi
```

#### En python

```
def maximum(tab):
    maxi = tab[0]
    for e in tab:
        if e > maxi:
            maxi = e
    return maxi

print(maximum([3,5,2,8,1]))
```

Ce programme affiche 8.

Voir dans PythonTutor