Parcours séquentiel d'un tableau

I – Algorithmique

Définition :

. . . . . .

. . . . . .

. . . . . .

Exercice 01 : Un algorithme créé par l'informaticien a besoin de n^3 opérations pour identifier la personne dans la base de données qui comporte n fiches. L'ordinateur peut effectuer 200 Giga opérations par seconde. Calculer les temps mis pour réaliser cette tâche pour 5000 fiches (5.10^3), 5 millions de fiches (5.10^6), 5 milliards de fiches (5.10^9).

Exercice 02 : Même question mais l'algorithme a besoin de n^2 opérations pour identifier la personne dans la base de données qui comporte n fiches. Par contre, l'ordinateur peut effectuer uniquement 30 Mega opérations par seconde.

Il existe des problèmes très complexes à résoudre de façon efficace : les **problèmes NP-complets**.

II – Algorithme de recherche linéaire (linear search)

But : trouver si un élément recherché existe bien dans un tableau. S'il existe, on fournit l'index de la première occurence trouvée. Sinon, il renvoie une réponse vide.

Principe : lecture séquentielle et progressive des différents éléments. On renvoie l'index du premier élément qui correspond. VIDE si aucun des éléments n'a correspondu à la recherche.

Description des entrées / sortie

ENTREES :
tableau : un tableau
x : l'élément qu'on tente de rechercher

SORTIE : une réponse vide (à définir) ou un nombre correspondant à l'index trouvé.

Description de l'algorithme

```
index ← 0
TANT QUE index < longueur et que tableau[index] est différent de x
    index ← index + 1
Fin TANT QUE
SI index < longueur
    Renvoyer index
Fin Si
Renvoyer VIDE
```

Exercice 03 : Appliquer l'algorithme à la main sur le tableau ci-dessous avec une recherche sur 1024. Ici, quelle est la condition qui provoque l'arrêt du TANT QUE : la valeur de la variable index ou le contenu qui est trouvé ?

Index	**0**	1	**2**	3	**4**	5	**6**
Elément	**13**	18	**89**	1024	**45**	-12	**18**

Exercice 04 : Appliquer l'algorithme à la main avec une recherche sur 1025. Vous devriez montrer qu'il renvoie VIDE. Quelle est la condition qui provoque l'arrêt du TANT QUE : la valeur de la variable index ou le contenu qui correspond à la valeur voulue ?

Exercice 05 : 8 apparaît deux fois dans le tableau. La valeur renvoyée par cet algorithme sur 18 donne :
A : **0** B : **1** C : **6** D : **(1,6)**

Exercice 06 : Implémenter cet algorithme en Python. Voir le prototype de la fonction sur le site.

REMARQUE : *il faut être très vigilant sur la condition initiale d'un TANT QUE. Il faut veiller à pouvoir rentrer dans la boucle au moins une première fois.*

III – VARIANT : Preuve de terminaison ou d'arrêt d'une boucle

Exercice de cours : Montrer la terminaison de l'algorithme suivant en montrant l'existence d'une condition du type **WHILE VARIANT > 0** et en montrant que ce variant est bien une **suite d'entiers positifs strictement décroissante**.

```
1  x = 3
2  while x < 100 :
3      x = x + 10
```

Exercice 07 : Montrer la terminaison de l'algorithme suivant en montrant l'existence d'une condition du type **WHILE VARIANT > 0** et en montrant que ce variant est bien une **suite d'entiers positifs strictement décroissante**.

```
1   def exercice(seuil) :
2       '''Fonction-exercice qui ne sert à rien
3       :: param seuil(int) :: un entier positif
4       :: return (int) :: un résultat
5       '''
6       x = 0
7       while 5*x < seuil :
8           x = x + 1
9       reponse = x
10      return reponse
```

Exercice 08 : Montrer la terminaison de l'algorithme précédent si on remplace l'une des lignes.

```
8              x = x - 1
```

Exercice 09 : Montrer la terminaison de l'algorithme de recherche linéaire en montrant l'existence d'une condition du type **WHILE VARIANT > 0** et en montrant que ce variant est bien une **suite d'entiers strictement décroissante**.

```
1   def trouverIndex(tableau, x) :
2       '''Fonction qui renvoie l'index de l'élément x dans le tableau. None si échec
3
4       :: param tableau(list) :: un tableau, même type que x
5       :: param x(au choix) :: l'élément qu'on cherche à trouver
6
7       :: exemples ::
8       >>> test = [10,20,30,40,50]
9       >>> trouverIndex(test, 20)
10      1
11      >>> trouverIndex(test, 60)
12
13      '''
14      index = 0
15      longueur = len(tableau)
16      while index < longueur and tableau[index] != x :
17          index = index + 1
18      if index < longueur :
19          return index
20      return None
```

IV – Complexité

Définition :

On cherchera souvent à comparer deux algorithmes faisant le même travail en cherchant à voir leur complexité pour répondre au pire des cas.

Dans le cas d'un algorithme de recherche dans un tableau, il s'agit du cas où l'élément cherché n'existe pas : on est alors obligé de lire les n éléments du tableau.

Exercice 10 : Dans le pire des cas, combien de fois la boucle va-t-elle s'effectuer si le tableau comporte 10, 100 ou 1000 éléments. A votre avis, a-t-on affaire à une complexité

- de type **constant** (en 1)
- de type **logarithmique** (en log n)
- de type **linéaire** (en n)
- de type **quadratique** (en n^2)
- de type **cubique** (en n^3)
- de type **exponentiel** (en 2^n) ?

V – Notations O et Θ (culture générale, pas de maîtrise réelle exigée)

Pour comparer les complexités des algorithmes entre eux, on utilise la notation suivante par exemple

Grand O : Permet de poser une borne supérieure du coût de l'algorithme.

$f(n) = O(n^2)$ indique que **f(n) est dominée** aux grandes valeurs de n par la fonction **n^2**.

Que veut dire dominée aux grandes valeurs ? **Le coût de f(n) est en n^2 ou moins**

L'algorithme réagit donc peut-être comme **n^2**, **n**, **$n^{1/2}$** ou moins encore.

On notera que si **$f(n) = O(n^2)$** alors on a également **$f(n) = O(n^3)$**, et donc **$f(n) = O(n^4)$** ...

Theta Θ : Permet de montrer l'odre de grandeur réel du coût de l'algorithme.

Exemple de cours 1 : si **$f(n) = 50\ n^2 - 10n + 5$** alors . . . . . .

Exemple de cours 2 : si **$f(n) = 50\ n - 10$** alors . . . . . .

Exercice 11 : Trouver le coût des algorithmes dont on vous donne l'expression mathématique du nombre d'opérations à effectuer en fonction du nombre n de données d'entrée. Barrer ensuite les expressions en O non valides.

$f(n) = 10\ n^4 + 3\ n^2 + 4000\ n$	f(n) = Θ ( )	f(n) = $O(n^4)$	f(n) = $O(n^3)$	f(n) = $O(n^2)$
$f(n) = 300\ n^3 + 9000\ n^2 + 10\ n$	f(n) = Θ ( )	f(n) = $O(n^4)$	f(n) = $O(n^3)$	f(n) = $O(n^2)$
$f(n) = 9000\ n^2 + 10\ n$	f(n) = Θ ( )	f(n) = $O(n^4)$	f(n) = $O(n^3)$	f(n) = $O(n^2)$

VI – Logarithme base 2 (hors NSI) – Voir la page sur le site

$Lg(2) = Log_2(2) = \log(\ ) =$ **$Lg(4) = Log_2(4) = \log(\ ) =$**

$Lg(8) = Log_2(8) = \log(\ ) =$ **$Lg(16) = Log_2(16) = \log(\ ) =$**

Remarque : équivalence entre (WHILE) et (FOR associé à RETURN)

```
1  def trouverIndex(tableau, x) :
2      '''Fonction qui renvoie l'index de l'élément x dans le tableau. None en cas d'échec de la recherche'''
3      for index in range( len(tableau) ) :
4          if tableau[index] == x :
5              return index
```