Algorithmique

Algorithmes opérants sur les tableaux unidimensionnels

Anicet E. T. Ebou, ediman.ebou@inphb.ci

01

Tableaux unidimensionnels

Préambule motivant

Écrire un algorithme qui enregistre 3 notes entières et calcule la moyenne de ces 3 notes.

Préambule motivant

Écrire un algorithme qui enregistre 3 notes entières et calcule la moyenne de ces 3 notes.

```
note1 = int(input(“Note 1: ”))

note2 = int(input(“Note 2: ”))

note3 = int(input(“Note 3: ”))

moy = (note1 + note2 + note3) / 3

print(“La moyenne est: “, moy)
```

Préambule motivant

Écrire un algorithme qui enregistre 100 notes entières et calcule la moyenne de ces 100 notes.

?

Importance et rôle des tableaux

1. Stocker une série de valeurs
2. Facilité de manipulation de plusieurs variables

Format des tableaux à une dimension

Un tableau est une structure ou type de données. Une variable de ce type contiendra:

- Plusieurs variables, disons ***n*** (n est alors la longueur de ce tableau);
- D'un même type, disons ***T.***

On appelle aussi les tableaux unidimensionnels vecteur ou vector ou array.

Format des tableaux à une dimension

Les ***n*** variables sont rangées de façon linéaire (en ligne) et repérées par un *nombre entier* compris entre **0** et **n-1** (dans la plupart des langages).

Ce nombre est appelé l'**index** ou l'**indice** et est mis la plupart du temps entre *crochets* après le nom de la variable de type tableau. Ici, on a représenté une variable **tab** de type tableau. Les **tab[i]** sont tous de type **T**.

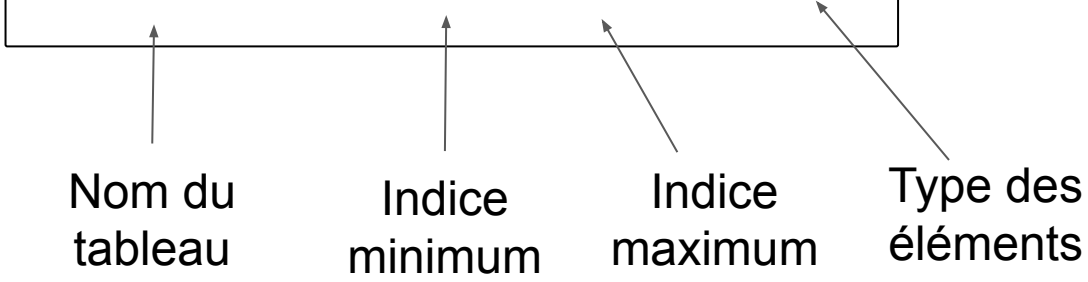
Contraintes sur les tableaux

- On rappelle que les éléments d'un même tableau sont d'un même type.
- De plus, la longueur du tableau est fixe. Une fois créé, un tableau ne peut pas changer de taille.

Syntaxe de déclaration en pseudo-code

```
Var:

    nom: TABLEAU[iMin..iMax] DE Type
```




```
# Exemple

Var:

    tab : TABLEAU [0..9] DE ENTIER
```

Syntaxe de déclaration en Python

Pour rappel (voir cours sur les variables), Python n'a pas de type tableau. Cependant, ce type de donnée est simulé à l'aide d'une liste qui correspond en Python à un tableau unidimensionnel.

```
# objet de type tableau

tab = [0] * taille
```

02

Manipulations élémentaires

Indexation d’un tableau unidimensionnel

Entrer une valeur dans un tableau par indexation

```
# Ecriture en pseudocode

tab[i] <- valeur

# Ecriture en python

tab[i] = valeur

# Lecture en pseudocode

Lire(tab[i])

# Lecture d’un entier Python

tab[i] = int(input())
```

Indexation d’un tableau unidimensionnel

Extraire/lire une valeur dans un tableau par indexation

```
# Extraction en pseudocode

v <- tab[i]

# Extraction en python

v = tab[i]

# Affichage en pseudocode

Ecrire(tab[i])

# Affichage en Python

print(tab[i])
```

Parcours d’un tableau unidimensionnel

Le principe est de « boucler sur les indices ». Puisqu’on connaît la longueur du tableau, on utilise le plus souvent une boucle POUR.

```
# Parcours en pseudocode
Pour i ← 0 à 9 Faire
     Ecrire('Entrer élément ', i)
     Lire(t[i])
FinPour
```

```
# Parcours en pseudocode
for i in range(10):
     print(f“Entrer élément {i}”)
     t[i] = int(input())
```

Travaux Pratiques

Écrire un algorithme qui enregistre 10 notes entières et calcule la moyenne de ces 10 notes.

03

Algorithmes de recherche

3.1

Algorithme de

Recherche d’un élément

Problème 1

Déterminer si un élément ***elt*** donné est dans un tableau unidimensionnel ***tab*** de taille ***n***?

Recherche d’un élément: algorithme 1

recherche1.py

```
1  is_in_tab = False
2  elt = input(“Veuillez entrer l’élément à rechercher: ”)
3  for i in range(len(tab)):
4       if elt == tab[i]:
5            is_in_tab = True
6  if is_in_tab:
7       print(“L’élément recherché est dans le tableau”)
8  else:
9       print(“L’élément recherché n’est dans le tableau”)
```

Recherche d’un élément: algorithme 2

recherche2.py

```
1  is_in_tab = False
2  elt = input(“Veuillez entrer l’élément à rechercher”)
3  for i in range(len(tab)):
4       if elt == tab[i]:
5            is_in_tab = True
6            break
7  if is_in_tab:
8       print(“L’élément recherché est dans le tableau”)
9  else:
10      print(“L’élément recherché n’est dans le tableau”)
```

Travaux Pratiques

Quelle est la différence entre les deux algorithmes précédents?

En déduire le rôle du mot-clé **break** en Python.

Recherche d’un élément

Problème 2

Déterminer la position d’un élément ***elt*** donné dans un tableau unidimensionnel ***tab*** de taille ***n***?

Recherche d’un élément: algorithme 1

recherche3.py

```
1   position = -1
2   elt = input(“Veuillez entrer l’élément à rechercher”)
3   for i in range(len(tab)):
4        if elt == tab[i]:
5             position = i
6        else:
7             position = -1
8   if position == -1:
9        print(“L’élément recherché n’est pas dans le tableau”)
10  else:
11       print(“L’élément recherché est à la position ”, position)
```

Recherche d’un élément: algorithme 2

recherche4.py

```
1  position = -1
2  elt = input(“Veuillez entrer l’élément à rechercher”)
3  for x in tab:
4       if elt == x:
5            position = i
6       else:
7            position = -1
8  if position == -1:
9       print(“L’élément recherché n’est pas dans le tableau”)
10 else:
11      print(“L’élément recherché est à la position ”, position)
```

Recherche du maximum

Problème 1

Trouver le maximum ***max*** d'un tableau unidimensionnel ***tab*** de taille ***n***.

Recherche du maximum

recherche6.py

```
1  max = tab[0]
2  for elt in tab:
3       if max < elt:
4            max = elt
5  print(“Le maximum est :“, max)
```

Recherche dichotomique: principe

Le tableau est nécessairement trié. Repérer le milieu du tableau, 3 cas possibles :

- Si x = t[milieu] alors x est trouvé et la recherche est terminée.
- Si x < t[milieu] alors chercher x dans le sous-tableau de gauche.
- Si x > t[milieu] alors chercher x dans le sous-tableau de droite.

Condition d’arrêt : t[milieu] = x ou dimension de sous tableau = 0

dicho.py

```
1   t = [1, 2, 3, 4, 5, 6, 7, 8, 9, 0]
2   v = int(input(‘Entrez la valeur à rechercher: “))
3   a = 0
4   b = len(t) - 1
5   while a <= b:
6        m = (a + b) // 2
7        if t[m] == v:
8             return True    # on a trouvé v
9        elif t[m] < v:
10            a = m + 1
11       else:
12            b = m - 1
13      return False              # on a a > b
```

Recherche dichotomique: principe

Les variables a et b représentent les bornes entre lesquelles on recherche v dans t. Autrement dit, avec les notations de Python, on recherche v dans t[a:b + 1].

Dans la version « classique » de l'algorithme, on effectue deux tests par itération, puisque l'on effectue le test d'égalité à chaque fois. On peut accélérer l'algorithme en ne faisant qu'un unique test d'égalité, en sortie de boucle.

dicho2.py

```
1   t = [1, 2, 3, 4, 5, 6, 7, 8, 9, 0]
2   v = int(input(‘Entrez la valeur à rechercher: “))
3   a = 0
4   b = len(t)
5   if b == 0: # il faut traîter le cas où la liste est vide
6        return False
7   while b > a + 1:
8        m = (a + b) // 2
9        if t[m] > v:
10            b = m
11       else:
12            a = m
13      return t[a] == v
```

Recherche de valeurs approchées d'une racine d'une équation algébrique

Pour déterminer une valeur approchée de solutions d'équations du type $f(x) = 0$, on peut utiliser trois méthodes : la méthode par dichotomie, la méthode de la sécante et la méthode de Newton.

Il arrive que certaines équations ne puissent pas être résolues algébriquement.

Recherche de valeurs approchées d'une racine d'une équation algébrique

Après avoir prouvé qu'elles admettent des solutions en utilisant, par exemple, le théorème des valeurs intermédiaires, il est alors utile d'avoir des méthodes pour déterminer une approximation numérique des solutions recherchées.

Méthode par dichotomie

- On considère une fonction f définie sur un intervalle I.
- On cherche à résoudre l'équation f(x) = 0 sur un intervalle [a ; b] après avoir prouvé que la fonction f est monotone et s'annule sur cet intervalle.
- On se fixe une précision e (par exemple à 10^{-2}).

Méthode par dichotomie

Pour cela, on utilise l'algorithme suivant:

- On partage l'intervalle [a ; b] en deux intervalles [a; m] et [m; b] avec m = (a + b) / 2.
- On choisit l'intervalle qui contient la solution pour cela, on calcule f(a) × f(m) :
  - si f(a) × f(m) ⩽ 0 cela signifie que f(a) et f(m) sont de signes contraires, donc la solution est dans l'intervalle [a ; m] ;
  - sinon la solution est dans l'intervalle [m ; b].

Méthode par dichotomie

On reprend l’étape 1 tant que (b – a) est supérieur à la précision e fixée. Pour cela, on remplace l’intervalle [a ; b] par celui qui contient la solution

Travaux Pratiques

Écrire un algorithme (pseudo-code et Python) qui permet de déterminer la première position d'un élément ***elt*** donné dans un tableau unidimensionnel ***tab*** de taille ***n***.

Écrire un algorithme (pseudo-code et Python) qui permet de déterminer toutes les positions d'un élément ***elt*** donné dans un tableau unidimensionnel ***tab*** de taille ***n***.

Travaux Pratiques (pour le mercredi 16 Novembre)

Écrire un algorithme (pseudo-code et Python) qui permet de déterminer le second maximum d'un tableau unidimensionnel ***tab*** de taille ***n***.

Écrire un algorithme (pseudo-code et Python) qui permet de déterminer le minimum d'un tableau unidimensionnel ***tab*** de taille ***n***.

Travaux Pratiques

On rappelle la définition d'une suite arithmétique:

$$\begin{cases} u_0 & = u_0 \\ u_{n+1} & = u_n + r \end{cases}$$

Écrire un programme (pseudo-code et Python) qui crée et affiche un vecteur ***tab*** de taille 10 pour y stocker les 10 premiers termes de la suite arithmétique dont le premier terme u_0 et la raison r sera fournie par l'utilisateur.

Travaux Pratiques

On rappelle la définition d'une suite géométrique $\begin{cases} u_0 & = u_0 \\ u_{n+1} & = u_n \times q \end{cases}$

Écrire un programme (pseudo-code et Python) qui crée et affiche un vecteur ***tab*** de taille 10 pour y stocker les 10 premiers termes de la suite géométrique dont le premier terme u_0 et la raison q sera fournie par l'utilisateur.

Travaux Pratiques

Écrire un programme (pseudo-code et Python) qui parcourt un tableau de nombres et qui remplace les nombres négatifs du tableau par 0.

Écrire un programme (pseudo-code et Python) qui parcourt un tableau de nombres et qui remplace les nombres impairs du tableau par 0.

Travaux Pratiques

On considère la fonction f définie sur [0 ; 1] par $f(x) = e^x - 2$.

Écrire en Python, en utilisant la méthode par dichotomie, un programme permettant de déterminer une valeur approchée à 0,1 près de la solution de l'équation $f(x) = 0$.