ALGORITHMIQUE · CHAPITRE 17 · CONDITIONS ET BOUCLES

Parcourir une suite de valeurs

Apprenez à parcourir une suite de valeurs, calculer sa somme et chercher son maximum en tenant compte des indices et du cas vide.

Par Ali ZAINOUL · Cours gratuit

Ce que vous allez apprendre

- Distinguer un indice et la valeur à cette position.
- Utiliser un accumulateur et un maximum provisoire.
- Traiter le vide avant d'accéder au premier élément.

Prérequis. Savoir écrire des conditions et des boucles POUR.

Une suite, deux résultats

Pour les valeurs `[4, 1, 7]`, nous voulons obtenir la somme 12 et le maximum 7. La suite contient des nombres réels finis ; l'exemple MATLAB utilise un vecteur ligne. Nous utilisons cette suite comme un support de parcours, sans étudier ici la conception d'une structure de données.

Valeur et position ont des rôles différents

`index` indique où lire, `value` contient ce que l'on vient de lire. Python commence les indices à 0, MATLAB à 1. Nous initialisons la somme à zéro, mais le maximum provisoire au premier élément : zéro ne serait pas un bon maximum de départ pour une suite entièrement négative.

Définir une suite, sa longueur et ses indices

Une **suite de valeurs** est une collection ordonnée : chaque élément possède une position. Sa **longueur** est le nombre d'éléments. Un **indice** est un numéro de position. Python et notre pseudo-code commencent à 0 ; MATLAB commence à 1. Pour trois valeurs, les indices sont donc 0, 1, 2 d'un côté et 1, 2, 3 de l'autre.

Dans `values = [4, 1, 7]`, les crochets construisent une liste Python ou un vecteur ligne MATLAB. Un **vecteur** est ici une suite numérique disposée sur une ligne ou une colonne. Python lit le premier élément avec `values[0]`, MATLAB avec `values(1)`. `len` et `numel` donnent le nombre d'éléments pour ces données.

Le **maximum** est la plus grande valeur présente. Initialiser sa recherche à zéro serait incorrect pour une suite telle que [-8, -3]. Nous partons donc du premier élément : le candidat initial est une valeur réellement présente. Mais une suite vide n'a pas de premier élément ; il faut traiter ce cas avant l'accès par indice.

La somme de notre suite vaut 12 et son maximum vaut 7. Après le premier élément, le total vaut 4 et le maximum vaut 4 ; après le second, le total vaut 5 et le maximum reste 4. Le troisième donne 12 et 7. Cette trace montre deux états distincts que la même boucle met à jour.

Protéger l’accès au premier élément

Pseudo-code

```
// example.pseudo
// summarize_values
values <- [4, 1, 7]
n <- length(values)
total <- 0
IF n == 0 THEN
    PRINT "No data"
ELSE
    maximum <- values[0]
    FOR index FROM 0 TO n - 1 INCLUSIVE DO
        value <- values[index]
        total <- total + value
        IF value > maximum THEN
            maximum <- value
        END IF
    END FOR
    PRINT total, maximum
END IF
```

Python

```python
# summarize_values.py
values = [4, 1, 7]
n = len(values)
total = 0
if n == 0:
    print("No data")
else:
    maximum = values[0]
    for index in range(n):
        value = values[index]
        total = total + value
        if value > maximum:
            maximum = value
    print(total, maximum)  # 12 7
```

MATLAB

```matlab
% summarize_values.m
values = [4, 1, 7];
n = numel(values);
total = 0;
if n == 0
    disp('No data');
else
    maximum = values(1);
    for index = 1:n
        value = values(index);
        total = total + value;
        if value > maximum
            maximum = value;
        end
    end
    disp([total, maximum]); % [12 7]
end
```

La condition sur la longueur précède l'accès au premier élément. Pour une suite vide, le programme affiche l'absence de données, sans tenter de produire un maximum. Sur une suite non vide, l'accès à chaque position contribue une seule fois à la somme.

Suivre les deux résultats partiels

Un seul parcours pour deux résultats

value	total après ajout	maximum après comparaison
4	4	4
1	5	4
7	12	7

Le maximum provisoire ne change pas à chaque tour : la comparaison avec 1 conserve 4. En revanche, la somme évolue à chaque valeur ajoutée. Les deux variables sont mises à jour selon des règles différentes, dans un même corps de boucle.

Vérifier les cas que l'on oublie

Essayez une suite d'une seule valeur : somme et maximum valent cet élément. Puis prévoyez le résultat pour `[-4, -1, -7]` : somme −12, maximum −1. Enfin, avec une suite vide, vérifiez que l'accès au premier élément n'est jamais atteint.

Adapter le parcours

▼ Comment compter les valeurs strictement positives ?

Initialiser count à zéro avant la boucle, puis augmenter count seulement lorsque value > 0. La valeur zéro n'est pas comptée. Le compteur est défini même si la suite est vide.

▼ **Pourquoi maximum ← 0 donnerait-il une mauvaise réponse pour [-4, -1, -7] ?**

Aucune valeur ne dépasserait zéro, qui resterait le maximum annoncé alors qu'il n'appartient même pas à la suite. Initialiser au premier élément relie le maximum provisoire à une valeur réellement observée.

Séparer données et méthode de parcours

Cette méthode montre comment lire une suite connue, accumuler et maintenir une propriété simple. Les choix entre listes, tableaux et autres structures, ainsi que leurs performances comparées, seront approfondis dans DSA.

À retenir - fiche mémo

Longueur

Le nombre d'éléments, donné ici par len ou numel.

Indices

Départ à 0 en pseudo-code/Python, à 1 en MATLAB.

Maximum

Pour une suite non vide, partez d'un élément réellement présent.

Pseudo-code

```
// example.pseudo
// first_element
// Pseudocode: zero-based indexing.
IF length(values) > 0 THEN
    maximum <- values[0]
END IF
```

Point de vigilance. Zéro n’est pas un bon maximum initial pour une suite contenant seulement des nombres négatifs.

Consulter la cheatsheet : Conditions et boucles · **Télécharger la fiche du thème en PDF**

Version web : https://www.clearcode.fr/courses/programming/algorithmic/control-flow/loops/sequences