Algorithmes de parcours séquentiel d'un tableau

Activité – Écriture des algorithmes et étude de leur coût

Pour chacun des trois algorithmes de parcours séquentiel au programme, cette fiche donne sa **spécification**, propose de l'écrire en **langage naturel** et en **Python** (parcours par valeur), puis d'étudier son **coût**.

Le **coût d'un algorithme** est le nombre d'opérations élémentaires (arithmétiques ou logiques) ainsi que d'affectations nécessaires à son exécution **dans le pire cas**. Le coût d'un algorithme dépend toujours de la taille des données d'entrée (ici la taille n du tableau).

Convention de comptage : on ne compte pas l'affectation de la variable de boucle (`elt`, qui prend successivement chaque valeur du tableau), ni le calcul de la longueur `len(tab)`, ni les instructions `renvoyer` (`return`).

I. Algorithme de recherche d'un élément

On veut écrire un algorithme qui cherche si une valeur `v` est présente ou non dans un tableau `tab`.

Spécification de l'algorithme

La **spécification** d'un algorithme précise de manière non ambigüe ce que doit faire un algorithme. En particulier, on y indique : le nom des données manipulées (entrées), le nom des données renvoyées (sorties), le rôle de l'algorithme ainsi que les hypothèses sur les entrées (précondition) et les sorties (postcondition). Un algorithme sera considéré comme *correct* s'il respecte cette spécification.

- **Entrée(s)** : un tableau `tab` de taille n et une valeur `v`
- **Sortie(s)** : un booléen
- **Rôle** : chercher si la valeur `v` est présente dans `tab`
- **Précondition(s)** : `v` est du même type que les éléments de `tab`
- **Postcondition(s)** : le booléen renvoyé vaut Vrai si `v` appartient à `tab`, et Faux sinon

Algorithme

✍ Écrivez l'algorithme en langage naturel et en Python (parcours par valeur).

Langage naturel	Python

Coût de l'algorithme

✍ **Question 1** : Dans quel(s) cas la boucle `pour chaque elt de tab` (`for elt in tab`) effectue-t-elle le plus grand nombre de tours ? Combien de tours fait-elle alors ? C'est le **pire cas**.

✍ **Question 2** : Dans le pire cas, combien de fois la comparaison `elt = v` (`elt == v`) est-elle évaluée ? Y a-t-il des affectations ? des opérations arithmétiques ?

✍ **Question 3** : Complétez le tableau suivant donnant le nombre d'opérations élémentaires dans le pire cas, puis déduisez-en le coût de l'algorithme.

Comparaisons	Affectations	Opérations arithmétiques

Remarque : parcours par indice. On peut aussi parcourir le tableau par indice :

```
def recherche_v2(v, tab):
    for i in range(len(tab)):
        if tab[i] == v:
            return True
    return False
```

Le coût est le même : n comparaisons dans le pire cas, donc un coût **linéaire**. Le parcours par indice devient **indispensable** si l'on veut renvoyer la *position* de `v` (et pas seulement Vrai ou Faux).

II. Algorithme de recherche du maximum

On veut écrire un algorithme qui renvoie la valeur maximale d'un tableau `tab`.

Spécification de l'algorithme

- **Entrée(s)** : un tableau `tab` de taille n
- **Sortie(s)** : un entier `maxi`
- **Rôle** : chercher l'élément maximal de `tab`
- **Précondition(s)** : `tab` est un tableau *non vide* d'entiers
- **Postcondition(s)** : `maxi` est un entier qui est l'élément maximal de `tab`

Algorithme

✍ Écrivez l'algorithme en langage naturel et en Python (parcours par valeur).

Langage naturel	Python

Coût de l'algorithme

✍ **Question 1** : Combien de tours effectue la boucle `pour chaque elt de tab` (`for elt in tab`) ? Combien de fois la comparaison `elt > maxi` est-elle évaluée ? Cela dépend-il des valeurs du tableau ?

✍ **Question 2** : L'affectation `maxi ← elt` (`maxi = elt`) n'est pas toujours effectuée. Donnez un exemple de tableau de taille $n = 5$ pour lequel elle est effectuée le plus grand nombre de fois : c'est le **pire cas**. Justifiez.

✍ **Question 3** : Au premier tour de boucle, `elt` vaut `tab[0]`. L'affectation `maxi ← elt` est-elle effectuée à ce tour ? Déduisez-en le nombre total d'affectations dans le pire cas (sans oublier l'initialisation `maxi ← tab[0]`).

✍ **Question 4** : Complétez le tableau suivant donnant le nombre d'opérations élémentaires dans le pire cas, puis déduisez-en le coût de l'algorithme.

Comparaisons	Affectations	Opérations arithmétiques

Remarque : parcours par indice. Le parcours par valeur compare inutilement `tab[0]` avec lui-même. Un parcours par indice permet de commencer au deuxième élément :

```
def recherche_max_v2(tab):
    maxi = tab[0]
    for i in range(1, len(tab)):
        if tab[i] > maxi:
            maxi = tab[i]
    return maxi
```

On effectue alors $n-1$ comparaisons et n affectations dans le pire cas, soit un coût de $2n-1$: le coût reste **linéaire**. Le parcours par indice devient **indispensable** si l'on veut renvoyer la *position* du maximum.

III. Algorithme de calcul d'une moyenne

On veut écrire un algorithme qui calcule la moyenne des nombres d'un tableau `tab`.

Spécification de l'algorithme

✍ **Question 1** : Proposez une spécification de cet algorithme.

- **Entrée(s)** :
- **Sortie(s)** :
- **Rôle** :
- **Précondition(s)** :
- **Postcondition(s)** :

Algorithme

✍ Écrivez l'algorithme en langage naturel et en Python (parcours par valeur).

Langage naturel	Python

Coût de l'algorithme

✍ **Question 2** : Le nombre d'opérations effectuées dépend-il des valeurs du tableau ? Que peut-on dire du pire cas ?

✍ **Question 3** : Combien d'additions et de divisions sont effectuées ? Combien d'affectations (sans oublier l'initialisation `somme ← 0`) ? Complétez le tableau puis déduisez-en le coût de l'algorithme.

Comparaisons	Affectations	Opérations arithmétiques

Remarque : parcours par indice. On peut aussi parcourir le tableau par indice :

```
def moyenne_v2(tab):
    somme = 0
    for i in range(len(tab)):
        somme = somme + tab[i]
    return somme / len(tab)
```

Le coût est le même ($2n + 2$), donc **linéaire**. La version par valeur, plus simple à écrire, peut être privilégiée.

Bilan

✍ Complétez le bilan suivant.

- La **spécification** d'un algorithme précise : .....
- Le **coût** d'un algorithme est calculé dans le ..... .
- Les trois algorithmes étudiés ont un coût de l'ordre de ..... (la taille du tableau) : on dit que leur coût est ..... .
- Le parcours **par valeur** est souvent plus simple à écrire, mais le parcours **par indice** est indispensable lorsque .....

Références :

- Cours du DIU EIL, Université de Nantes, C. JERMANN.

Germain BECKER & Sébastien POINT, Lycée Mounier, ANGERS – Licence CC BY-NC-SA 4.0