Mémo n. 3

Algorithmes classiques sur les tableaux

I32 Preuves et Analyses d'algorithmes

P. Véron

Dans ce mémo nous traitons uniquement le cas des tableaux mono-dimensionnels.

1. Tableaux de dimension 1

Définition : *Un tableau est un ensemble de variables de même type, désignées par un même nom, et distingués les unes des autres par leur numéro (appelé aussi indice).*

Déclaration :
- T tableau de n <type>, ou
- T tableau de n <type> indexé de ℓ à m ($\ell \leqslant m, m - \ell + 1 = n$).

Dans le premier cas, le tableau est automatiquement indexé de 1 à n.

Exemple : Les deux déclarations suivantes correspondent à n variables de type entiers,
- T tableau de n entiers,
- T tableau de n entiers indexé de 3 à $n + 2$

Dans le premier cas les variables ont pour nom $T[1], T[2], \ldots, T[n]$, dans l'autre cas elles ont pour nom $T[3], T[4], \ldots, T[n + 2]$.

2. Représentation en C

En C les tableaux sont indexés à partir de la valeur 0. La déclaration générale est de la forme : $<$ type $>$ $tab[n]$, où tab est l'identificateur du tableau et n est une expression constante.
Exemple : int $T[10]$ déclare 10 variables entières $T[0], \ldots, T[9]$.
Le tableau est stocké dans $n\times$(taille du type) octets consécutifs en mémoire. L'identificateur tab est stocké à une certaine adresse mémoire dont le contenu est initialisée avec l'adresse du premier élément du tableau. On dit que la variable tab est un point d'entrée en mémoire pour le tableau tab.

Adresse	Contenu	
0	...	
⋮	...	
10	160	T
⋮	⋮	
⋮	⋮	
160	??	$T[0]$
⋮	⋮	⋮
⋮	⋮	⋮
...	...	$T[n-1]$

Fig. 1 Représentation d'un tableau en mémoire.

$$
\begin{aligned}
\text{adresse } T[0] &= \text{valeur}(T) \\
\text{adresse } T[1] &= \text{adresse(T[0])} + \text{taille du type} \\
&= \text{valeur}(T) + \text{taille du type} \\
&\vdots \\
\text{adresse } T[j] &= \text{valeur}(T) + j\times \text{ taille du type}
\end{aligned}
$$

Dans l'exemple ci-dessus, quelle est l'adresse de $T[1]$, si T est un tableau d'entiers ? de réels ? de caractères ?

3. Recherche d'un élément

But : Déterminer la position du 1er élément qui vérifie une propriété P donnée.

$$P : \mathcal{A} \to \{\text{VRAI}, \text{FAUX}\}$$
$$x \mapsto P(x) = \begin{cases} \text{VRAI} & \text{si } x \text{ vérifie } P, \\ \text{FAUX} & \text{sinon.} \end{cases}$$

A est le domaine de définition du tableau T ($\forall i, T[i] \in \mathcal{A}$).

```
1  Algo recherche
2  données
3     T : tableau de n ...
4     k : entier
5     oncontinue : booléen
6     début
7        lire(T)
8        oncontinue ← non(P(T[1]))
9        k ← 2
10       tant que (k ⩽ n) et oncontinue faire
11                oncontinue ← non(P(T[k]))
12                k ← k + 1
13       fintq
14       si non(oncontinue) alors
15          ⋮     (T[k − 1] est l'élément recherché)
16       finsi
17    fin
```

Preuve : On utilise l'invariant suivant pour démontrer l'exactitude de cet algorithme

$\{\forall i, 1 \leqslant i < k-1, P(T[i]) = \text{FAUX}\}$ et $\{\text{oncontinue} \Leftrightarrow \text{non}(P(T[k-1]))\}$ et $\{0 \leqslant k \leqslant n+1\}$

4. Parcours

But : Appliquer un même traitement à tous les éléments d'un tableau (ou un traitement impliquant tous les éléments du tableau).

```
1  Algo parcours
2  données
3     T : tableau de n ...
4     i : entier
5     {variables nécessaires au traitement}
6     début
7        lire(T)
8        (initialistaion du traitement)
9        i ← 1
10       tant que i ⩽ n faire
11                (traiter l'élément T[i])
12                i ← i + 1
13       fintq
14       (afficher éventuellement le résultat du traitement)
15    fin
```

Preuve : On utilise l'invariant suivant pour démontrer l'exactitude de cet algorithme

$$\{\forall j, 1 \leqslant j < i, T[j] \text{ a été traité et } 0 \leqslant i \leqslant n+1\}$$

5. Parcours partiel

But : Appliquer un même traitement à une sous-séquence du tableau délimitée à sa droite par le premier élément vérifiant une propriété P.

```
1  Algo parcours partiel
2  données
3     T : tableau de n ...
4     k : entier
5     oncontinue : booléen
6     {variables nécessaires au traitement}
7     début
8        lire(T)
9        (initialistaion du traitement)
10       oncontinue ← non(P(T[1]))
11       k ← 1
12       tant que (k < n) et oncontinue faire
13                (traiter l'élément T[k])
14                k ← k + 1
15                oncontinue ← non(P(T[k]))
16       fintq
17       si oncontinue alors
18                          (traiter T[n])
19       finsi
20       (cette dernière étape est inutile
21       s'il existe un élément du tableau vérifiant P)
22    fin
```

Preuve : On utilise l'invariant suivant pour démontrer l'exactitude de cet algorithme

$$\{\forall i, 1 \leqslant i < k, P(T[i]) = \text{FAUX}\} \text{ et } \{\text{oncontinue} \Leftrightarrow \text{non}(P(T[k]))\} \text{ et}$$
$$\{\forall i, 1 \leqslant i < k, T[i] \text{ a été traité}\} \text{ et } \{0 \leqslant k \leqslant n+1\}$$

6. Parcours sélectif

But : Appliquer un même traitement uniquement aux éléments du tableau vérifiant une propriété P.

```
1  Algo parcours sélectif
2  données
3     T : tableau de n ...
4     i : entier
5     {variables nécessaires au traitement}
6     début
7        lire(T)
8        (initialistaion du traitement)
9        i ← 1
10       tant que i ⩽ n faire
11                si P(T[i]) alors
12                                  (traiter T[i])
13                finsi
14                i ← i + 1
15       fintq
16    fin
```

Preuve : On utilise l'invariant suivant pour démontrer l'exactitude de cet algorithme

$$\{\forall j, 1 \leqslant j < i, T[j] \text{ a été traité } \Leftrightarrow P(T[j])\} \text{ et } \{0 \leqslant i \leqslant n+1\}$$

7. Exemples

• *Recherche du premier élément positif dans un tableau T d'entiers.* Il s'agit de l'algorithme classique de recherche avec :

$$\begin{array}{rcccl} P & : & \mathbb{Z} & \to & \{\text{VRAI}, \text{FAUX}\} \\ & & x & \mapsto & P(x) = \begin{cases} \text{VRAI} & \text{si } x \geqslant 0, \\ \text{FAUX} & \text{sinon.} \end{cases} \end{array}$$

```
1  Algo recherche
2  données
3     T : tableau de n entiers
4     k : entier
5     oncontinue : booléen
6     début
7        lire(T)
8        oncontinue ← (T[1] < 0)
9        k ← 2
10       tant que (k ⩽ n) et oncontinue faire
11                oncontinue ← (T[k] < 0)
12                k ← k + 1
13       fintq
14       si non(oncontinue) alors
15          ⋮      (T[k − 1] est l'élément recherché)
16       finsi
17    fin
```

• *Somme de tous les éléments du tableau.* Il s'agit de l'algorithme classique de parcours. Ici le traitement correspond à remettre à jour la variable S qui contient à chaque instant la somme courante calculée. L'initialisation du traitement revient à mettre à zéro cette valeur au début de l'algorithme.

```
1  Algo parcours
2  données
3     T : tableau de n entiers
4     i : entier
5     S : entier
6     début
7        lire(T)
8        S ← 0
9        i ← 1
10       tant que i ⩽ n faire
11                S ← S + i
12                i ← i + 1
13       fintq
14       ecrire(S)
15    fin
```

• *Calcul de la somme de tous les éléments situés avant le premier élément positif (dont on est certain de l'existence).* Il s'agit de l'algorithme classique de parcours partiel. La propriété P est la même que celle du premier exemple.

```
1  Algo parcours partiel
2  données
3     T : tableau de n entiers
4     k : entier
5     oncontinue : booléen
6     S : entier
7     début
8        lire(T)
9        S ← 0
10       oncontinue ← (T[1] < 0)
11       k ← 1
12       tant que (k < n) et oncontinue faire
13                S ← S + T[k]
14                k ← k + 1
15                oncontinue ← (T[k] < 0)
16       fintq
17       ecrire(S)
18    fin
```

• *Calcul de la somme des éléments pairs d'un tableau.* Il s'agit de l'algorithme classique du parcours sélectif avec :

$$\begin{array}{rcccl} P & : & \mathbb{Z} & \to & \{\text{VRAI}, \text{FAUX}\} \\ & & x & \mapsto & P(x) = \begin{cases} \text{VRAI} & \text{si } x \bmod 2 = 0, \\ \text{FAUX} & \text{sinon.} \end{cases} \end{array}$$

```
1  Algo parcours sélectif
2  données
3     T : tableau de n entiers
4     i : entier
5     S : entier
6     début
7        lire(T)
8        S ← 0
9        i ← 1
10       tant que i ⩽ n faire
11                si (T[i] mod 2 = 0) alors
12                                        S ← S + T[i]
13                finsi
14                i ← i + 1
15       fintq
16       ecrire(S)
17    fin
```