Méthodes de conception de programmes
Devoir 1 : Preuve de programme

Alexandre Gobeaux[a], Louis Navarre[a], Gilles Peiffer[a]

[a]*École Polytechnique, Université catholique de Louvain, Place de l'Université 1, 1348 Ottignies-Louvain-la-Neuve, Belgique*

Abstract

Ce document donne un algorithme pour résoudre le problème 2SUM ainsi qu'une preuve de correction totale pour celui-ci, par rapport aux spécifications définies.

1. Description du problème et de la solution

Le problème à résoudre est celui d'une séquence d'entiers triée a dans laquelle il faut déterminer si oui ou non deux d'entre-eux (potentiellement deux fois le même) ont une somme égale à un entier prédéfini s. Si oui, alors le programme doit renvoyer « `true` », ainsi que les indices des deux entiers qui satisfont la condition, sinon il renvoie « `false` », et la valeur des indices n'a pas d'importance.

Notre algorithme fonctionne en temps linéaire ($\mathcal{O}(n)$), en utilisant deux pointeurs : le premier commence au premier élément du tableau, alors que le deuxième commence à la fin. On évalue leur somme à chaque itération, tant que le premier pointeur est plus bas que le second :

— si celle-ci est plus petite que s, on avance d'une unité le premier pointeur ;
— si celle-ci est plus grande que s, on recule d'une unité le deuxième pointeur et
— si celle-ci est égale à s, on retourne les valeurs ainsi que la valeur « `true` ».

L'algorithme peut donc se terminer de deux façons : soit il trouve une paire d'indices qui satisfait la condition, passant donc dans la troisième possibilité ci-dessus, soit les pointeurs se croisent, ce qui signifie que la séquence ne contient pas de paire satisfaisant la condition.

2. Spécification formelle

2.1. Précondition

La précondition est que la séquence soit triée. Formellement, on demande que

$$\forall k, \ell \mid 0 \geq k \leq \ell < |a| :: a[k] \leq a[l]\,.$$

2.2. Modifie

L'algorithme ne modifie rien.

2.3. Postcondition

La première postcondition est, informellement, que si la valeur booléenne est vraie, alors i et j contiennent les bonnes valeurs d'indices. La seconde postcondition est que si la valeur booléenne est fausse, alors la somme de toute paire est différente de s et que si la somme est différente de s pour toute paire, alors la valeur booléenne retournée est fausse. Formellement (en appelant `found` la valeur booléenne),

$$\begin{aligned} \texttt{found} &\implies (0 \leq i \leq j < |a|) \wedge (a[i] + a[j] = s)\,, \\ \wedge \neg\texttt{found} &\iff \forall m, n \mid 0 \leq m \leq n < |a| :: a[m] + a[n] \neq s\,. \end{aligned}$$

3. Implémentation

L'implémentation de l'algorithme donné précédemment en Dafny est la suivante.

Email addresses: `alexandre.gobeaux@student.uclouvain.be` (Alexandre Gobeaux), `louis.navarre@student.uclouvain.be` (Louis Navarre), `gilles.peiffer@student.uclouvain.be` (Gilles Peiffer)

```
method find_sum(a: seq<int>, s: int) returns (found: bool, i: int, j: int)
requires sorted(a)
ensures found ⟹ (0 ≤ i ≤ j < |a| ∧ a[i] + a[j] = s)
ensures ¬found ⟺ (∀ m, n | 0 ≤ m ≤ n < |a| • a[m] + a[n] ≠ s)
{
  i, j := 0, |a| - 1;
  while i ≤ j
  invariant 0 ≤ i ≤ j+1 ≤ |a|
  invariant ∀ p, x | 0 ≤ p < i ∧ 0 ≤ x < |a| • a[p] + a[x] ≠ s
  invariant ∀ q, x | j < q < |a| ∧ 0 ≤ x < |a| • a[x] + a[q] ≠ s
  {
    var m;
    m := a[i] + a[j];
    if m > s {
      j := j - 1;
    } else if m < s {
      i := i + 1;
    } else {
      found := true;
      return;
    }
  }
  found := false;
}
predicate sorted(a: seq<int>)
{
  ∀ k, l | 0 ≤ k ≤ l < |a| • a[k] ≤ a[l]
}
```

4. Graphe d'exécution

Le graphe d'exécution pour l'algorithme ci-dessus est donné sur la FIGURE 1. Afin de faciliter les preuves pour les différents chemins simples, la FIGURE 2 reprend ceux-ci de façon plus lisible.

Références

[1] https://www.robtex.com/dns-lookup/www.aliexpress.com, accessed on December 1, 2018.

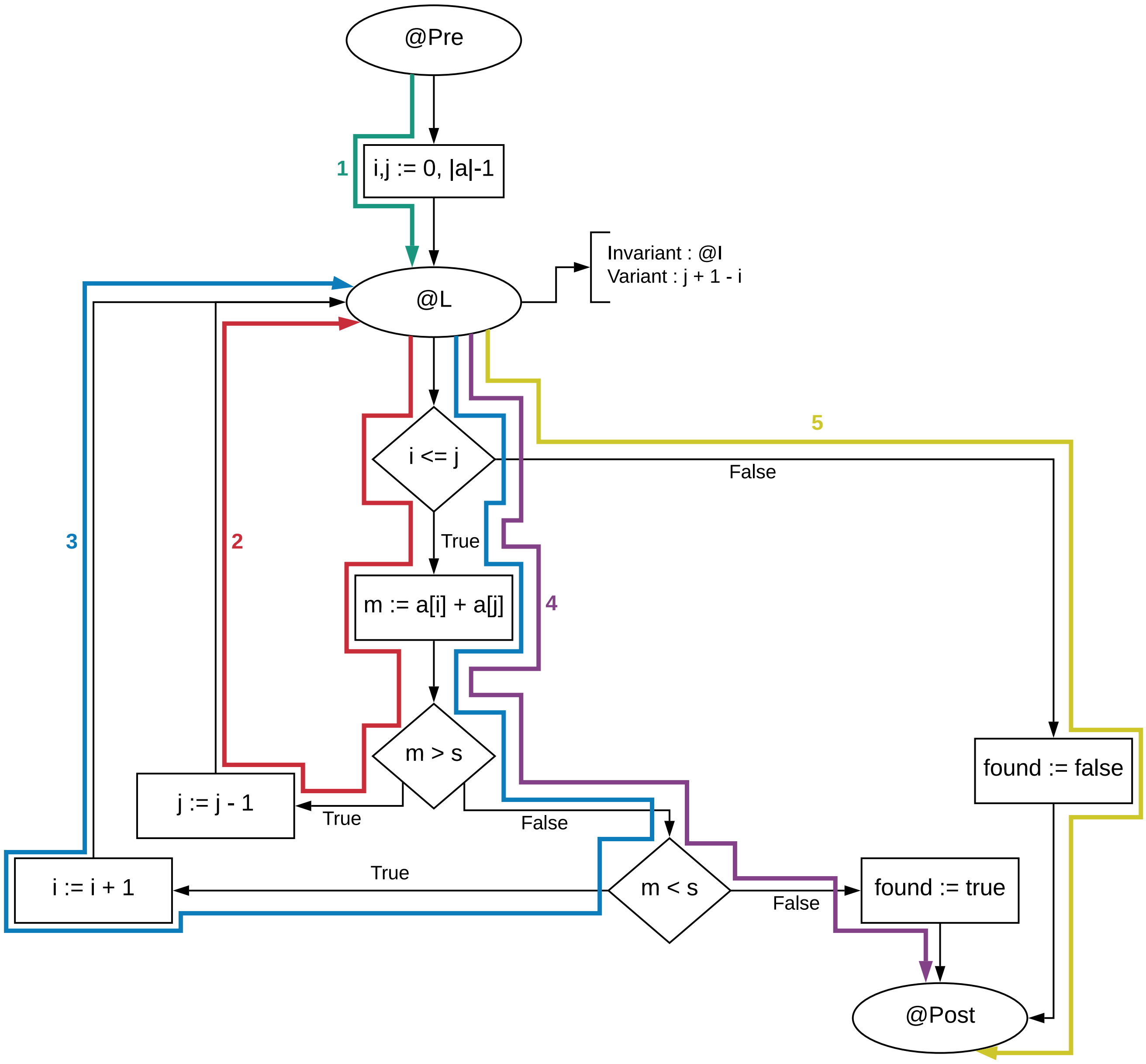

Figure 1: Le graphe d'exécution complet pour le programme donné.

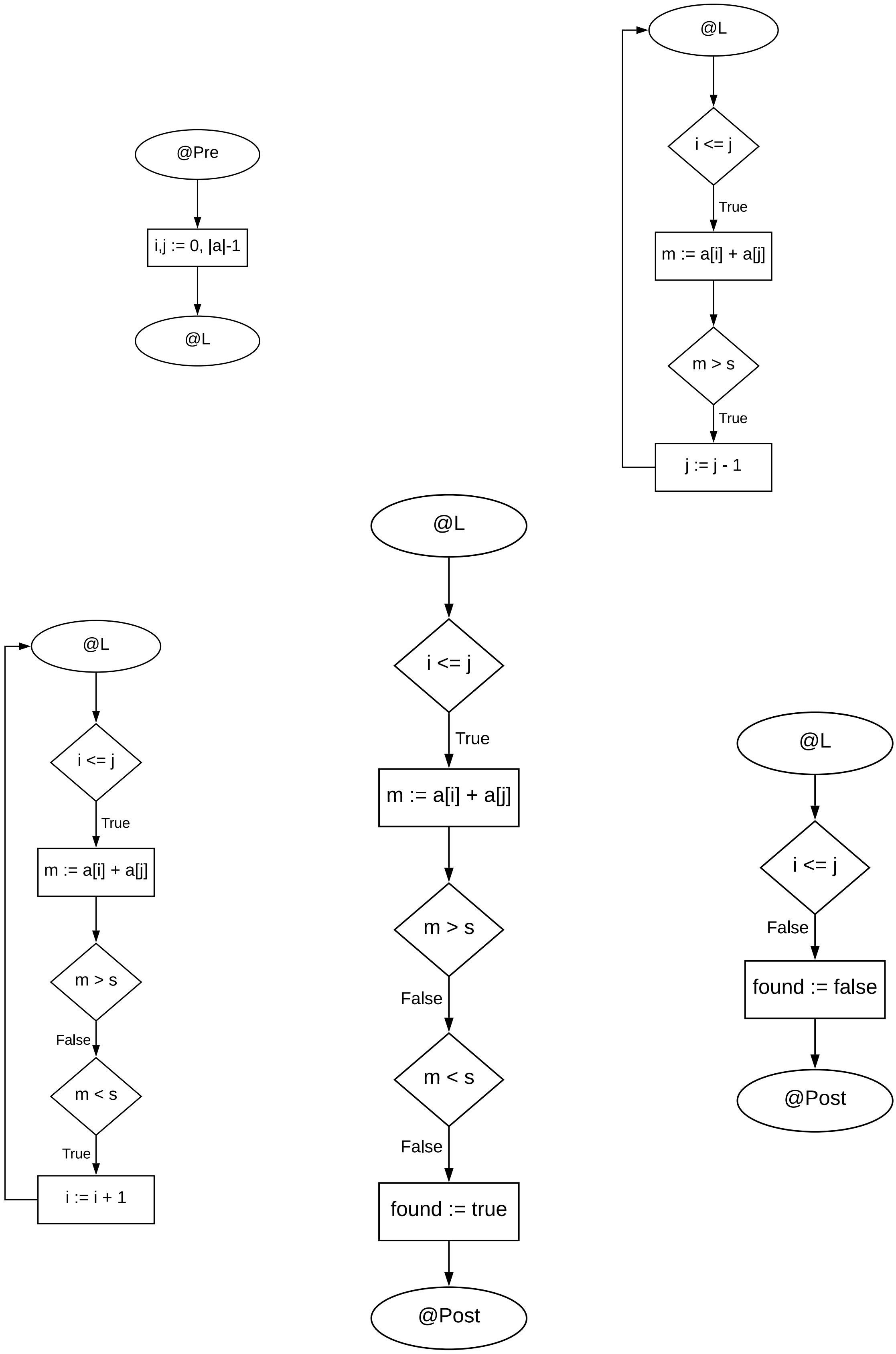

Figure 2: Représentations plus propres des différents chemins simples.