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

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

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, chaque chemin simple est redessiné à coté de sa preuve de correction.

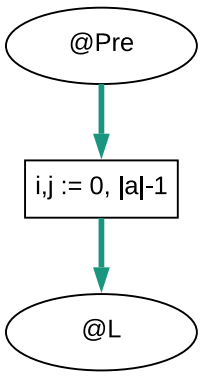
5. Preuve de correction

Avant de prouver la correction des différents chemins simples, définissons quelques prédicats :

$$\texttt{sorted}(a) \equiv \forall k, \ell \mid 0 \leq k \leq \ell < |a| :: a[k] \leq a[\ell] \tag{Pre}$$

$$\begin{array}{l} 0 \leq i \leq j+1 \leq |a| \\ \wedge \forall p, x \mid 0 \leq p < i \wedge 0 \leq x < |a| :: a[p] + a[x] \neq s \\ \wedge \forall q, x \mid j < q < |a| \wedge 0 \leq x < |a| :: a[x] + a[q] \neq s \end{array} \tag{I}$$

5.1. Précondition jusqu'à l'entrée de la boucle



@Pre → @I.

$\left[\text{Pre}\right]$

$\left[0 \leq 0 \leq |a| - 1 + 1 \leq |a|\right.$

$\wedge \forall p, x \mid 0 \leq p < 0 \wedge 0 \leq x < |a| :: a[p] + a[x] \neq s$

$\left.\wedge \forall q, x \mid |a| - 1 < q < |a| \wedge 0 \leq x < |a| :: a[x] + a[q] \neq s\right]$

```
i, j := 0, |a| − 1;
```

$\left[\text{I}\right]$

Afin d'affirmer la correction de ce chemin simple, on doit encore prouver que le deuxième prédicat est bien une conséquence de la précondition.

Démonstration. On peut simplifier ce deuxième prédicat pour montrer qu'il est en fait équivalent à $\top$. Il est donc toujours impliqué par la précondition. □

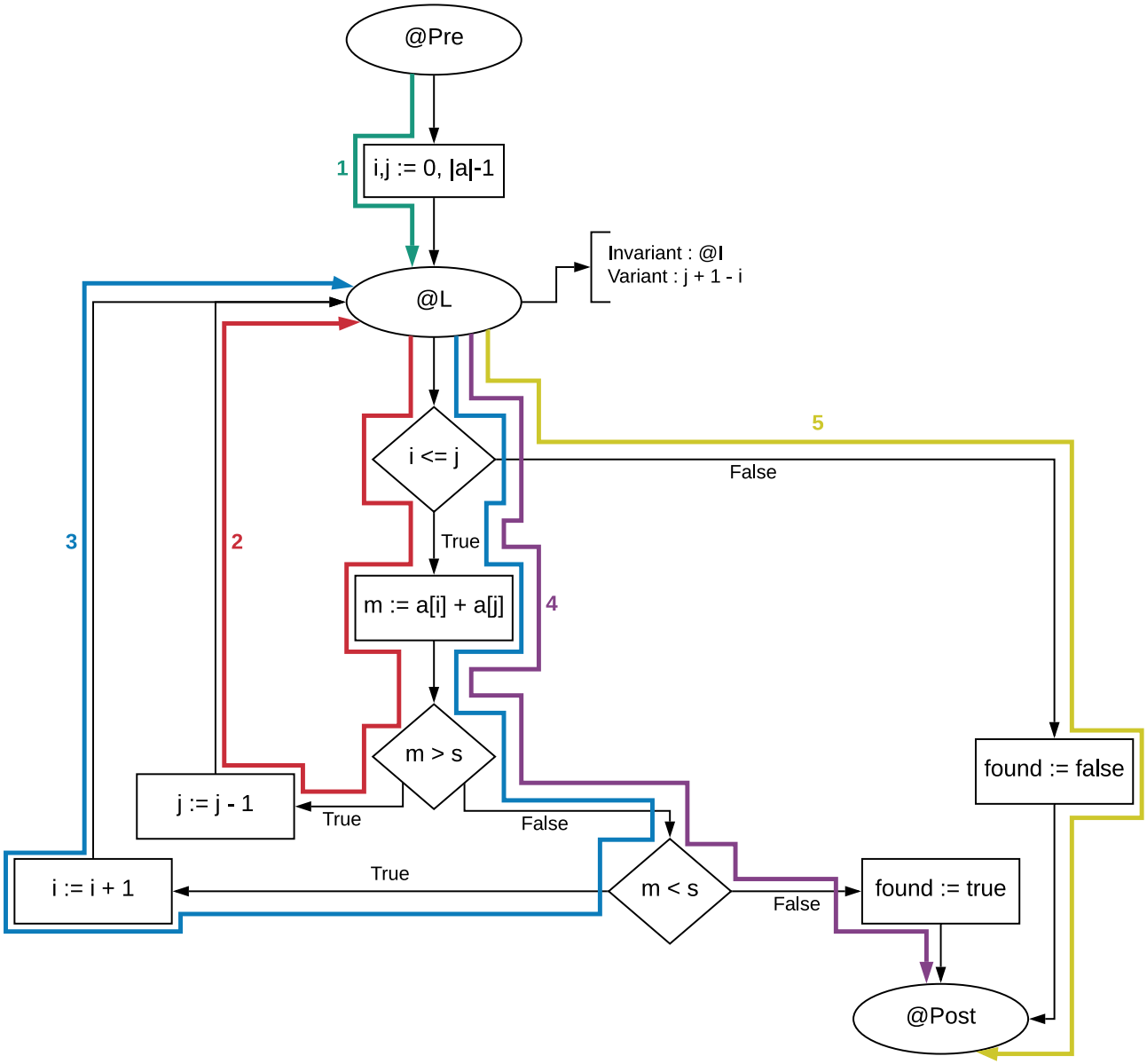


Figure 1: Le graphe d'exécution complet pour le programme donné.

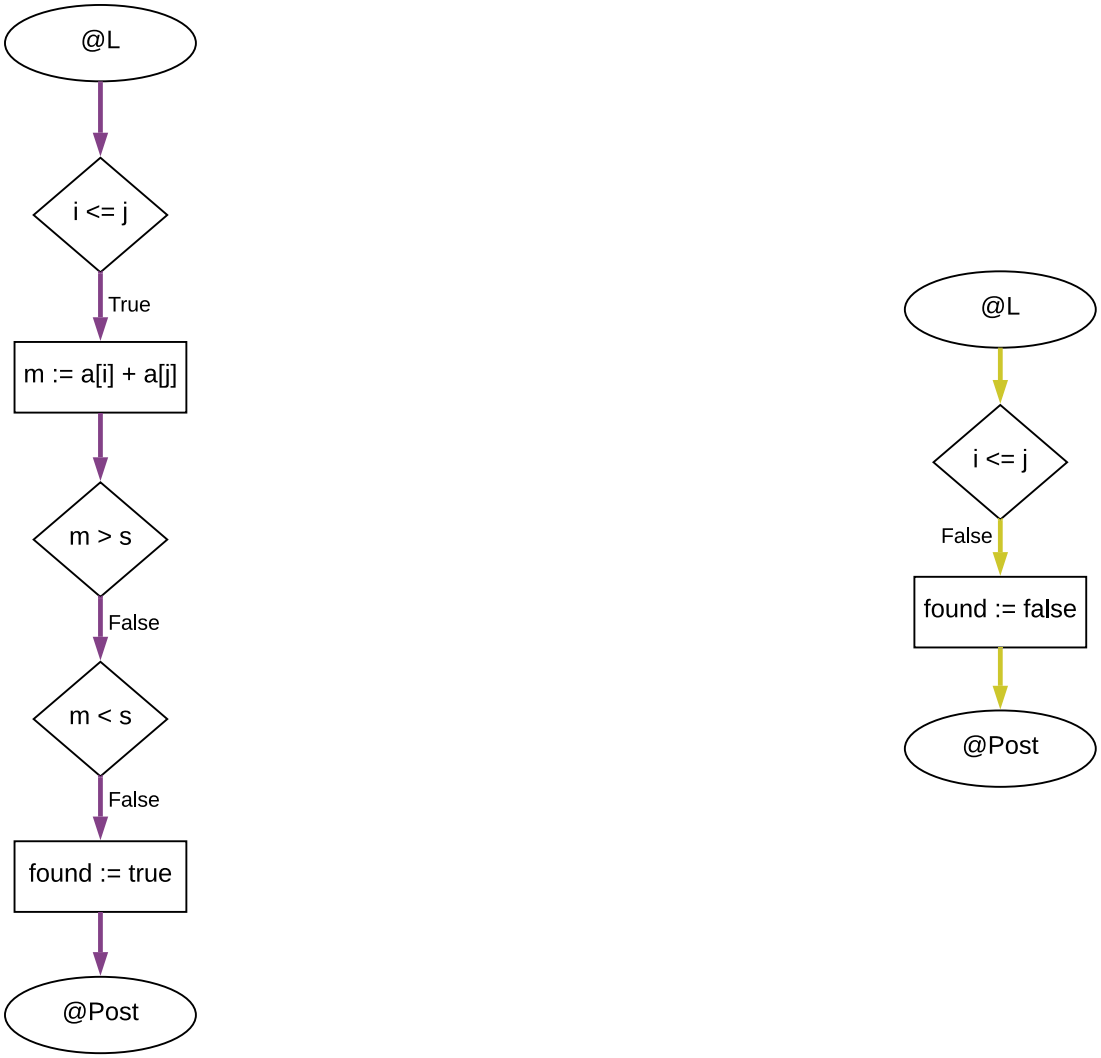
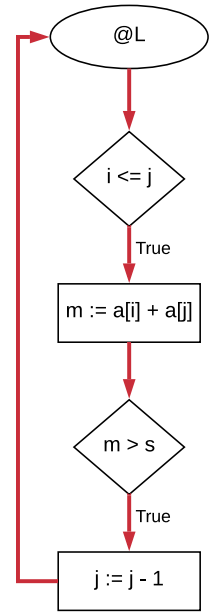


Figure 2: Représentations plus propres des différents chemins simples.



@I $\rightarrow$ @I pour $m > s$.

$\Big[\mathrm{I}\Big]$

$\Big[i \leq j \implies \big(a[i] + a[j] > s \implies \mathrm{I}_2\big)\Big]$

`assume i ≤ j;`

$\Big[a[i] + a[j] > s \implies \mathrm{I}_2\Big]$

`m := a[i] + a[j];`

$\Big[m > s \implies \mathrm{I}_2\Big]$

`assume m > s;`

$\Big[\mathrm{I}_2\Big] \equiv \Big[0 \leq i \leq j \leq |a|$

$\wedge \forall p, x \mid 0 \leq p < i \wedge 0 \leq x < |a| :: a[p] + a[x] \neq s$

$\wedge \forall q, x \mid j - 1 < q < |a| \wedge 0 \leq x < |a| :: a[x] + a[q] \neq s\Big]$

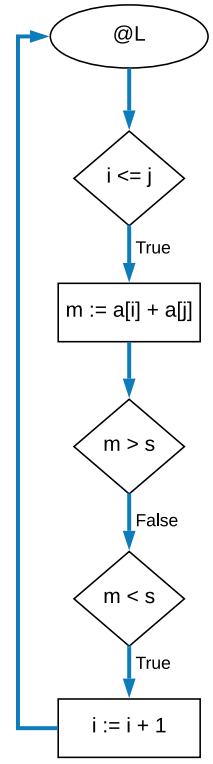
`j := j - 1;`

$\Big[\mathrm{I}\Big]$

Pour montrer que l'invariant implique bien le deuxième prédicat, on doit prouver que

$$\Big(\mathrm{I} \implies \Big[i \leq j \implies \big(a[i] + a[j] > s \implies \mathrm{I}_2\big)\Big]\Big)$$
$$\equiv \Big[\mathrm{I} \wedge i \leq j \wedge a[i] + a[j] > s\Big] \implies \mathrm{I}_2\,.$$

Démonstration. Pour ne pas trop surcharger les équations, on procède partie par partie. Dans l'invariant I, la première partie de la conjonction devient $0 \leq i < j \leq |a|$, car $(i \leq j + 1) \wedge (i \leq j) \equiv i \leq j$ et $j + 1 \leq |a| \implies j \leq |a|$. La deuxième partie de la conjonction est exactement la même dans I et I_2, et ne nécessite donc pas de développement. Pour la troisième partie, elle est identique sauf dans le cas $q = j$. Dans ce cas, comme on a également que $a[i] + a[j] > s$, et que le tableau est trié, le seul moyen d'obtenir $a[i] + a[j] = s$ avec $a[j]$ fixé serait de prendre un élément $a[p] < a[i]$, avec donc $0 \leq p < i$, comme le tableau est trié. Or, par la deuxième partie de la conjonction de l'invariant, on sait que $\forall p, x \mid 0 \leq p < i \wedge 0 \leq x < |a| :: a[p] + a[x] \neq s$. Dans notre cas de figure, il suffit de prendre $x = j$ pour prouver qu'on n'aura jamais $a[p] + a[j] = s$ pour $0 \leq 0p < i$. □



@I → @I pour $m < s$.

$\left[\mathrm{I}\right]$

$\left[i \leq j \implies \left(a[i] + a[j] < s \implies \mathrm{I}_3\right)\right]$

`assume i ≤ j;`

$\left[a[i] + a[j] < s \implies \mathrm{I}_3\right]$

`m := a[i] + a[j];`

$\left[\left(\neg(m > s) \implies (m < s \implies \mathrm{I}_3)\right) \equiv \left(m < s \implies \mathrm{I}_3\right)\right]$

`assume ¬(m > s);`

$\left[m < s \implies \mathrm{I}_3\right]$

`assume m < s;`

$\left[\mathrm{I}_3\right] \equiv \Big[0 \leq i + 1 \leq j + 1 \leq |a|$

$\wedge \forall p, x \mid 0 \leq p < i + 1 \wedge 0 \leq x < |a| :: a[p] + a[x] \neq s$

$\wedge \forall q, x \mid j < q < |a| \wedge 0 \leq x < |a| :: a[x] + a[q] \neq s\Big]$

`i := i +1;`

$\left[\mathrm{I}\right]$

Pour montrer que l'invariant implique bien le deuxième prédicat, on doit prouver que

$$\left(\mathrm{I} \implies \left[i \leq j \implies \left(a[i] + a[j] < s \implies \mathrm{I}_3\right)\right]\right)$$
$$\equiv \left[\mathrm{I} \wedge i \leq j \wedge a[i] + a[j] < s\right] \implies \mathrm{I}_3\,.$$

Démonstration. Pour ne pas trop surcharger les équations, on procède partie par partie. Dans l'invariant I, la première partie de la conjonction devient $0 \leq i < j \leq |a|$, car $(i \leq j + 1) \wedge (i \leq j) \equiv i \leq j$ et $j + 1 \leq |a| \implies j \leq |a|$. La troisième partie de la conjonction est exactement la même dans I et I_3, et ne nécessite donc pas de développement. Pour la deuxième partie, elle est identique sauf dans le cas $p = i$. Dans ce cas, comme on a également que $a[i] + a[j] < s$, et que le tableau est trié, le seul moyen d'obtenir $a[i] + a[j] = s$ avec $a[i]$ fixé serait de prendre un élément $a[q] > a[j]$, avec donc $j < p < |a|$, comme le tableau est trié. Or, par la troisième partie de la conjonction de l'invariant, on sait que $\forall q, x \mid j < p < |a| \wedge 0 \leq x < |a| :: a[x] + a[q] \neq s$. Dans notre cas de figure, il suffit de prendre $x = i$ pour prouver qu'on n'aura jamais $a[i] + a[q] = s$ pour $j < q < |a|$. □

Références

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