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 ou équivalent au 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 un pointeur dépasse l'autre, ce qui signifie que la séquence ne contient pas de paire satisfaisant la condition.

2. Spécifications formelles

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
  decreases j − i
  {
    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. Le point de coupe se trouve à l'entrée de la boucle.

5. Preuve de correction partielle

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

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

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

$$\begin{array}{l} \texttt{found} \implies 0 \leq i \leq j < |a| \wedge a[i] + a[j] = s \\ \wedge\, \neg\texttt{found} \iff \forall m, n \mid 0 \leq m \leq n < |a| :: a[m] + a[n] \neq s \end{array} \tag{Post}$$

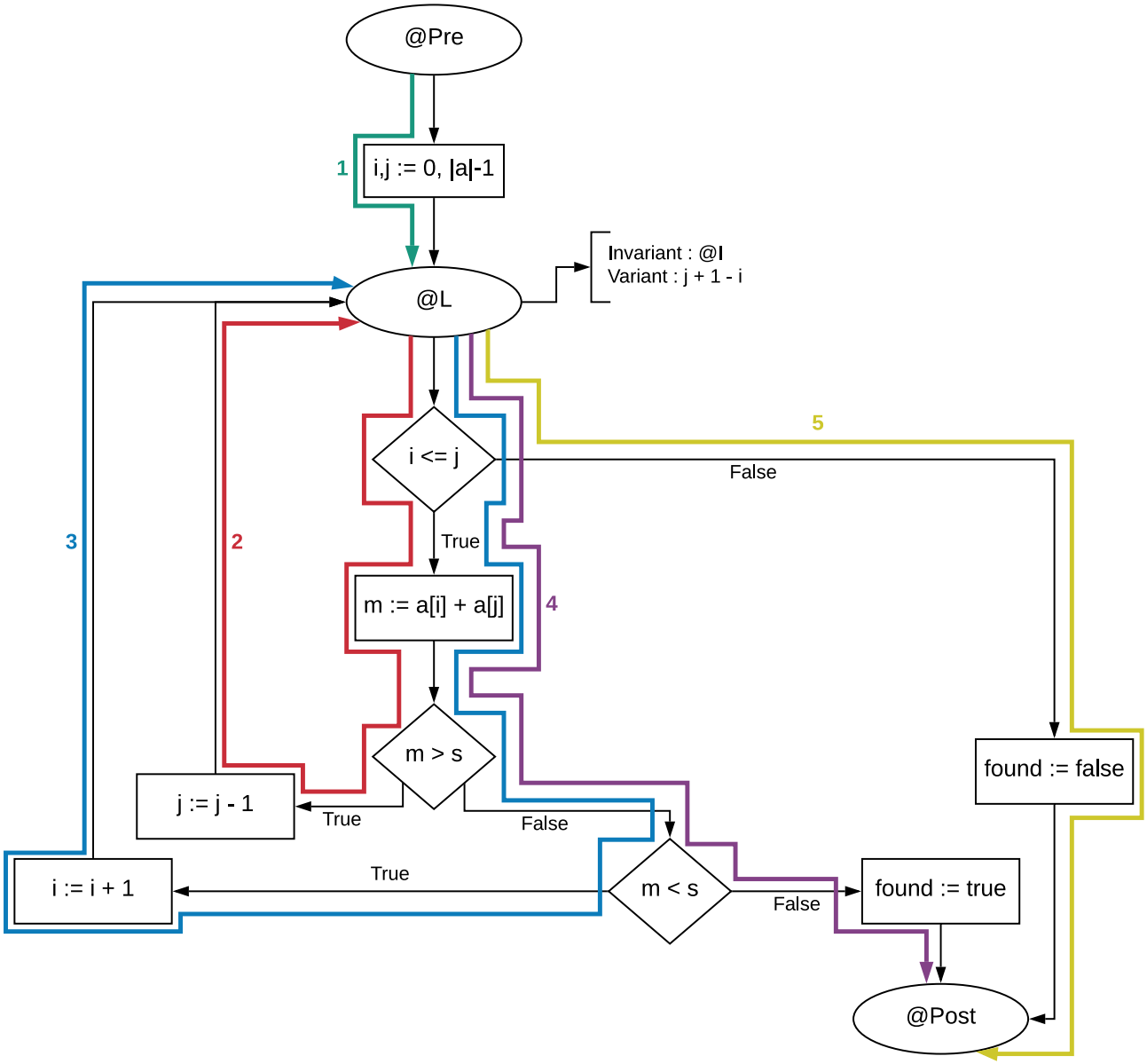


Figure 1: Le graphe d'exécution complet pour le programme donné.

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

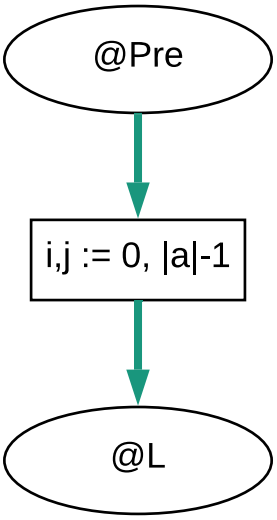


FIGURE 2: @Pre → @I.

Démonstration.

$$\Big[\text{Pre}\Big]$$
$$\Big[0 \leq 0 \leq |a| - 1 + 1 \leq |a|$$
$$\wedge\, \forall p, x \mid 0 \leq p < 0 \wedge 0 \leq x < |a| :: a[p] + a[x] \neq s$$
$$\wedge\, \forall q, x \mid |a| - 1 < q < |a| \wedge 0 \leq x < |a| :: a[x] + a[q] \neq s\Big]$$
`i, j := 0, |a| - 1;`
$$\Big[\text{I}\Big]$$

Afin d'affirmer la correction de ce chemin simple, on doit encore prouver que le deuxième prédicat est bien une conséquence de la précondition. On peut simplifier ce deuxième prédicat pour montrer qu'il est en fait équivalent à $\top$. Il est donc toujours impliqué par la précondition. □

5.2. Boucle dans le cas $m > s$

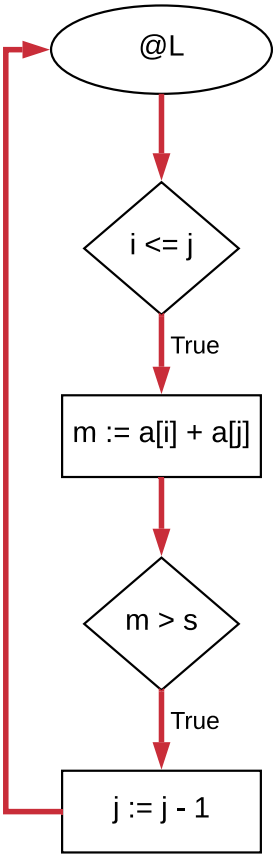


FIGURE 3: @I → @I pour $m > s$.

Démonstration.

$$\Big[\text{I}\Big]$$
$$\Big[i \leq j \implies (a[i] + a[j] > s \implies \text{I}_2)\Big]$$
`assume i ≤ j;`
$$\Big[a[i] + a[j] > s \implies \text{I}_2\Big]$$
`m := a[i] + a[j];`
$$\Big[m > s \implies \text{I}_2\Big]$$
`assume m > s;`
$$\Big[\text{I}_2\Big] \equiv \Big[0 \leq i \leq j \leq |a|$$
$$\wedge\, \forall p, x \mid 0 \leq p < i \wedge 0 \leq x < |a| :: a[p] + a[x] \neq s$$
$$\wedge\, \forall q, x \mid j - 1 < q < |a| \wedge 0 \leq x < |a| :: a[x] + a[q] \neq s\Big]$$
`j := j - 1;`
$$\Big[\text{I}\Big]$$

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

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

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 $0 \leq p < i$. □

5.3. Boucle dans le cas $m < s$

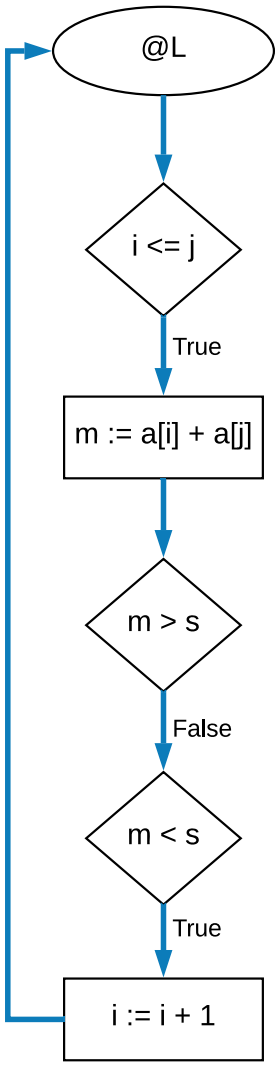


FIGURE 4: @I → @I pour $m < s$.

Démonstration.

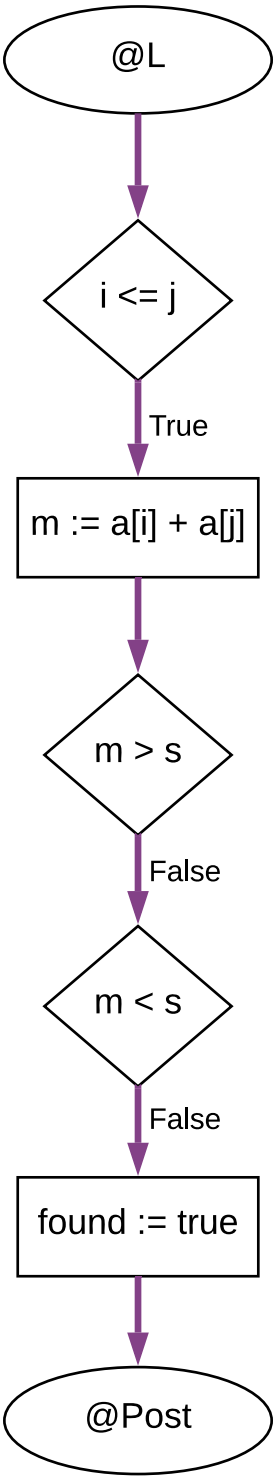
$$\Big[\mathrm{I}\Big]$$
$$\Big[i \le j \implies \big(a[i] + a[j] < s \implies \mathrm{I}_3\big)\Big]$$
`assume i ≤ j;`
$$\Big[a[i] + a[j] < s \implies \mathrm{I}_3\Big]$$
`m := a[i] + a[j];`
$$\Big[\big(\neg(m > s) \implies (m < s \implies \mathrm{I}_3)\big)\Big] \equiv \Big[\big(m < s \implies \mathrm{I}_3\big)\Big]$$
`assume ¬(m > s);`
$$\Big[m < s \implies \mathrm{I}_3\Big]$$
`assume m < s;`
$$\Big[\mathrm{I}_3\Big] \equiv \Big[0 \le i+1 \le j+1 \le |a|$$
$$\wedge\, \forall p, x \mid 0 \le p < i+1 \wedge 0 \le x < |a| :: a[p] + a[x] \ne s$$
$$\wedge\, \forall q, x \mid j < q < |a| \wedge 0 \le x < |a| :: a[x] + a[q] \ne s\Big]$$
`i := i +1;`
$$\Big[\mathrm{I}\Big]$$

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

$$\Big(\mathrm{I} \implies \Big[i \le j \implies \big(a[i] + a[j] < s \implies \mathrm{I}_3\big)\Big]\Big)$$
$$\equiv \Big[\mathrm{I} \wedge i \le j \wedge a[i] + a[j] < s\Big] \implies \mathrm{I}_3\,.$$

Pour ne pas trop surcharger les équations, on procède partie par partie. Dans l'invariant I, la première partie de la conjonction devient $0 \le i < j \le |a|$, car $(i \le j + 1) \wedge (i \le j) \equiv i \le j$ et $j + 1 \le |a| \implies j \le |a|$. 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 $j < q < |a|$, comme le tableau est trié. Or, par la troisième partie de la conjonction de l'invariant, on sait que $\forall q, x \mid j < q < |a| \wedge 0 \le x < |a| :: a[x] + a[q] \ne s$. Dans notre cas de figure, il suffit de prendre $x = i$ pour prouver qu'on n'aura jamais $a[i] + a[q] = s$ pour $j < q < |a|$. □

5.4. Entrée de la boucle jusqu'à la sortie dans le cas $a[i] + a[j] = s$



Démonstration.

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

$\Big[i \le j \implies \big(a[i] + a[j] = s \implies \mathrm{Post}_2\big)\Big]$

`assume i ≤ j;`

$\Big[a[i] + a[j] = s \implies \mathrm{Post}_2\Big]$

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

$\Big[\neg(m > s) \implies \big(\neg(m < s) \implies \mathrm{Post}_2\big)\Big] \equiv \Big[(m = s) \implies \mathrm{Post}_2\Big]$

`assume ¬(m>s);`

$\Big[\neg(m < s) \implies \mathrm{Post}_2\Big]$

`assume ¬(m<s);`

$\Big[\mathrm{Post}_2\Big] \equiv \Big[\top \implies 0 \le i \le j < |a| \land a[i] + a[j] = s$

$\land \neg\top \iff \forall m, n \mid 0 \le m \le n < |a| :: a[m] + a[n] \ne s\Big]$

`found := true;`

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

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

$$\begin{aligned}&\Big(\mathrm{I} \implies \Big[i \le j \implies \big(a[i] + a[j] = s \implies \mathrm{Post}_2\big)\Big]\Big)\\ \equiv&\Big[\mathrm{I} \land i \le j \land a[i] + a[j] = s\Big] \implies \mathrm{Post}_2\,.\end{aligned}$$

Pour ne pas trop surcharger les équations, on procède partie par partie. Dans l'invariant I, la première partie de la conjonction devient $0 \le i < j \le |a|$, car $(i \le j + 1) \land (i \le j) \equiv i \le j$ et $j + 1 \le |a| \implies j \le |a|$. Le prédicat Post_2 peut être réécrit comme

$$\begin{aligned}\mathrm{Post}_2 \equiv&\Big(0 \le i \le j < |a| \land a[i] + a[j] = s\Big)\\ &\land \Big(\exists m, n \mid 0 \le m \le n < |a| :: a[m] + a[n] = s\Big)\,.\end{aligned}$$

On voit donc que la condition $a[i] + a[j] = s$ réapparaît également. Finalement, on prend le cas $m = i$, $n = j$, ce qui représente la deuxième partie de Post_2. On a donc bien prouvé que l'invariant est une conséquence de la plus faible précondition de ce tableau. □

FIGURE 5: @I → @Post pour $a[i] + a[j] = s$.

5.5. Entrée de la boucle jusqu'à la sortie dans le cas $a[i] + a[j] \neq s$

Démonstration.

$$\Big[\mathrm{I}\Big]$$
$$\Big[\neg(i \leq j) \implies \mathrm{Post}_3\Big] \equiv \Big[i > j \implies \mathrm{Post}_3\Big]$$
`assume ¬(i ≤ j);`
$$\Big[\mathrm{Post}_3\Big] \equiv \Big[\bot \implies 0 \leq i \leq j < |a| \wedge a[i] + a[j] = s$$
$$\wedge \neg\bot \iff \forall m, n \mid 0 \leq m \leq n < |a| :: a[m] + a[n] \neq s\Big]$$
`found := false;`
$$\Big[\mathrm{Post}\Big]$$

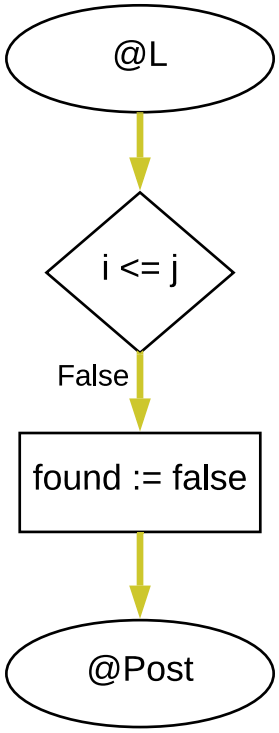


FIGURE 6: @I → @Post pour $a[i] + a[j] \neq s$.

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

$$\Big(\mathrm{I} \implies \Big[i > j \implies \mathrm{Post}_3\Big]\Big)$$
$$\equiv \Big[\mathrm{I} \wedge i > j\Big] \implies \mathrm{Post}_3\,.$$

Pour ne pas trop surcharger les équations, on procède partie par partie. Dans l'invariant I, la première partie de la conjonction devient $j < i \leq j + 1 \leq |a|$, et donc $i = j + 1$. On peut alors réécrire l'invariant (qui est en réalité devenu $\mathrm{I} \wedge i > j$) comme

$$\begin{aligned}
&-1 \leq j < i = j + 1 \leq |a| \\
&\wedge \forall p, x \mid 0 \leq p < j + 1 \wedge 0 \leq x < |a| :: a[p] + a[x] \neq s \\
&\wedge \forall q, x \mid j < q < |a| \wedge 0 \leq x < |a| :: a[x] + a[q] \neq s \\
\equiv\ &-1 \leq j < i = j + 1 \leq |a| \\
&\wedge \forall x, y \mid 0 \leq y < |a| \wedge 0 \leq x < |a| :: a[x] + a[y] \neq s\,.
\end{aligned}$$

On réécrit aussi le prédicat Post_3, comme suit :

$$\mathrm{Post}_3 \equiv \Big(\forall m, n \mid 0 \leq m \leq n < |a| :: a[m] + a[n] \neq s\Big)\,.$$

On remarque bien que $\mathrm{I} \wedge i > j \implies \mathrm{Post}_3$, ce qui conclut la preuve. □

6. Preuve de terminaison

Afin de montrer que le programme se termine, il faut montrer que le variant diminue à chaque itération. On choisit ici le variant $j - i$. Pour faire ceci, on distingue trois cas :

— la valeur de $a[i] + a[j] = m > s$;
— la valeur de $a[i] + a[j] = m < s$;
— la valeur de $a[i] + a[j] = m = s$.

Dans les deux premiers cas, on démontre relativement facilement que le variant diminue d'une unité à chaque itération, alors que dans le troisième cas, le programme se termine immédiatement, peu importe la valeur du variant. On se limitera donc à la preuve des deux premiers cas. Comme montré au cours, il faudra donc pour chaque chemin simple $@L'_i S @L'_j$ prouver $[P_i \wedge V_i = v_0]\ S\ [V_j < v_0]$.

6.1. Évolution du variant dans le cas $m > s$

Dans ce cas, montré à la FIGURE 3, l'évolution du variant au cours de l'exécution est ajoutée aux tableau d'assertions déjà donné.

$$\Big[j - i = V_0 \wedge \mathrm{I}\Big]$$
$$\Big[i \leq j \implies \big(a[i] + a[j] > s \implies j - 1 - i < V_0\big)\Big]$$
`assume i ≤ j;`
$$\Big[a[i] + a[j] > s \implies j - 1 - i < V_0\Big]$$
`m ≔ a[i] + a[j];`
$$\Big[m > s \implies j - 1 - i < V_0\Big]$$
`assume m > s;`
$$\Big[j - 1 - i < V_0\Big]$$
`j ≔ j − 1;`
$$\Big[j - i < V_0\Big]$$

On a bien $j - 1 - i < j - i$. On trouve donc bien que le variant à la sortie de la boucle est strictement plus petit que le variant en entrée. Comme on sait que la valeur minimale du variant est atteinte lorsque $i = j+1 \implies j - i = -1$ (on a en fait $j - i \geq -1$), le programme doit se terminer après un nombre fini d'étapes.

6.2. Évolution du variant dans le cas $m < s$

Dans ce cas, montré à la FIGURE 3, l'évolution du variant au cours de l'exécution est ajoutée aux tableau d'assertions déjà donné.

$$\Big[j - i = V_0 \wedge \mathrm{I}\Big]$$
$$\Big[i \leq j \implies \big(a[i] + a[j] < s \implies j - 1 - i < V_0\big)\Big]$$
`assume i ≤ j;`
$$\Big[a[i] + a[j] < s \implies j - 1 - i < V_0\Big]$$
`m ≔ a[i] + a[j];`
$$\Big[\big(\neg(m > s) \implies (m < s \implies j - 1 - i < V_0)\big)\Big] \equiv \Big[m < s \implies j - 1 - i < V_0\Big]$$
`assume ¬(m > s);`
$$\Big[m < s \implies j - 1 - i < V_0\Big]$$
`assume m < s;`
$$\Big[j - 1 - i < V_0\Big]$$
`j ≔ j − 1;`
$$\Big[j - i < V_0\Big]$$

On a bien $j - 1 - i < j - i$. On trouve donc bien que le variant à la sortie de la boucle est strictement plus petit que le variant en entrée. Comme on sait que la valeur minimale du variant est atteinte lorsque $i = j+1 \implies j - i = -1$ (on a en fait $j - i \geq -1$), le programme doit se terminer après un nombre fini d'étapes.

7. Conclusion

Comme nous avons prouvé à la SECTION 5 que les différents chemins d'exécution sont corrects, et que nous avons prouvé à la SECTION 6 que le programme se termine après un nombre fini d'étapes, on a ainsi prouvé la correction totale de l'algorithme.

C'est grâce à la garantie de diminution du variant d'ailleurs qu'il est possible d'affirmer que le temps d'exécution de l'algorithme est linéaire en la taille de la séquence d'entiers.