LINFO1122 - Méthodes de conception de programme : DEVOIR N°1 2023

Auteurs :
Jérôme Lechat
Arthur Louette
Arthur Fraipont

NOMA :
5035-18-00
5108-21-00
7738-20-00

1 Introduction

L'objectif de ce premier devoir nous a été donné sur moodle et nous allons l'expliquer simplement ici : Il est demandé d'écrire une fonction find-sum qui reçoit en argument un tableau trié a et un entier s et qui doit trouver 2 éléments distincts du tableau a dont la somme vaut s. Si il existe 2 éléments tels que leur somme vaut s, alors on y retourne un tuple composé des 2 indices des 2 éléments.
La complexité de l'algorithme utilisé n'est pas importante.

2 Solution

Vous trouverez ci-dessous notre algorithme implémenté en langage Java.

```
public static int[] find_Sum(int[] a, int b) {
    int start = 0;
    int end = a.length - 1;
    while (start < end) {
        int result = a[start] + a[end];
        if (result == b) {
            int[] results = {start, end};
            return results;
        } else if (result < b) {
            start++;
        } else {
            end--;
        }
    }
    return new int[]{-1, -1};
}
```

Tout d'abord, nous initialisons 2 variables start et end, qui valent respectivement 0 et la longueur de la liste passée en paramètre - 1. Ensuite, nous mettons une boucle while qui continue d'itérer tant que start est plus petit que end.

Dans cette boucle while, on va créer une nouvelle variable result qui vaut la valeur de a à l'indice start + la valeur de a à l'indice end. Si la nouvelle variable result vaut la valeur de l'entier s, alors on retourne un tuple composé de start et end.

Si result est plus grand que la valeur recherchée s, on incrémente de + 1 pour start, et on reste dans la boucle. Si inversément on voit que result est plus petit que s, on décrémente de - 1 end tout en restant dans la boucle. Une fois sortie de la boucle while, c'est que nous n'avons pas trouvé de valeur répondant à nos critères, on retourne alors un simple tuple (-1,-1).

3 Spécifications

3.1 Preconditions

a est non null

$$@Pre : a \neq null$$

a est trié par ordre croissant

$$@Pre : \forall i, j \, (0 \leq i < j < a.length \Rightarrow a[i] \leq a[j])$$

3.2 Postconditions

Si une paire d'indices (i, j) est retournée :

$$@Post : (result[0] \neq -1 \wedge result[1] \neq -1) \Rightarrow (0 \leq result[0] < result[1] < a.length \wedge a[result[0]] + a[result[1]] = b)$$

Si la valeur -1 est retournée, alors aucune paire n'a été trouvée :

$$@Post : (result[0] = -1 \wedge result[1] = -1) \Rightarrow (\forall i, j\ (0 \leq i < j < a.length \Rightarrow a[i] + a[j] \neq b))$$

3.3 Invariants

Invariant de boucle : À chaque itération, la somme des éléments à `start` et `end` est comparée à b :

$$\forall i, j\ (0 \leq i < start < j \leq end \Rightarrow a[i] + a[j] \neq b)$$

Tableau trié : À chaque itération, le tableau reste trié :

$$\forall i, j\ (0 \leq i < j < a.length \Rightarrow a[i] \leq a[j])$$

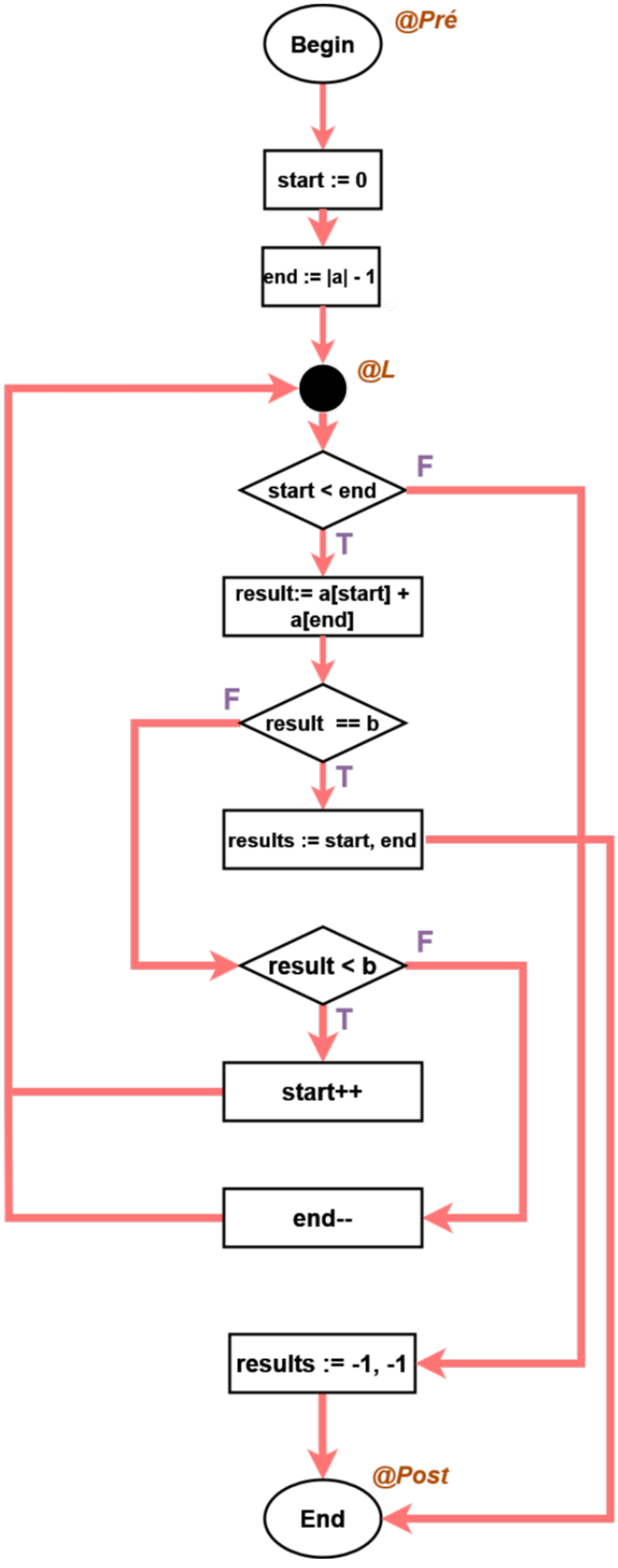
3.4 Variants

Variante de boucle : La différence entre `end` et `start` est une variante décroissante à chaque itération :

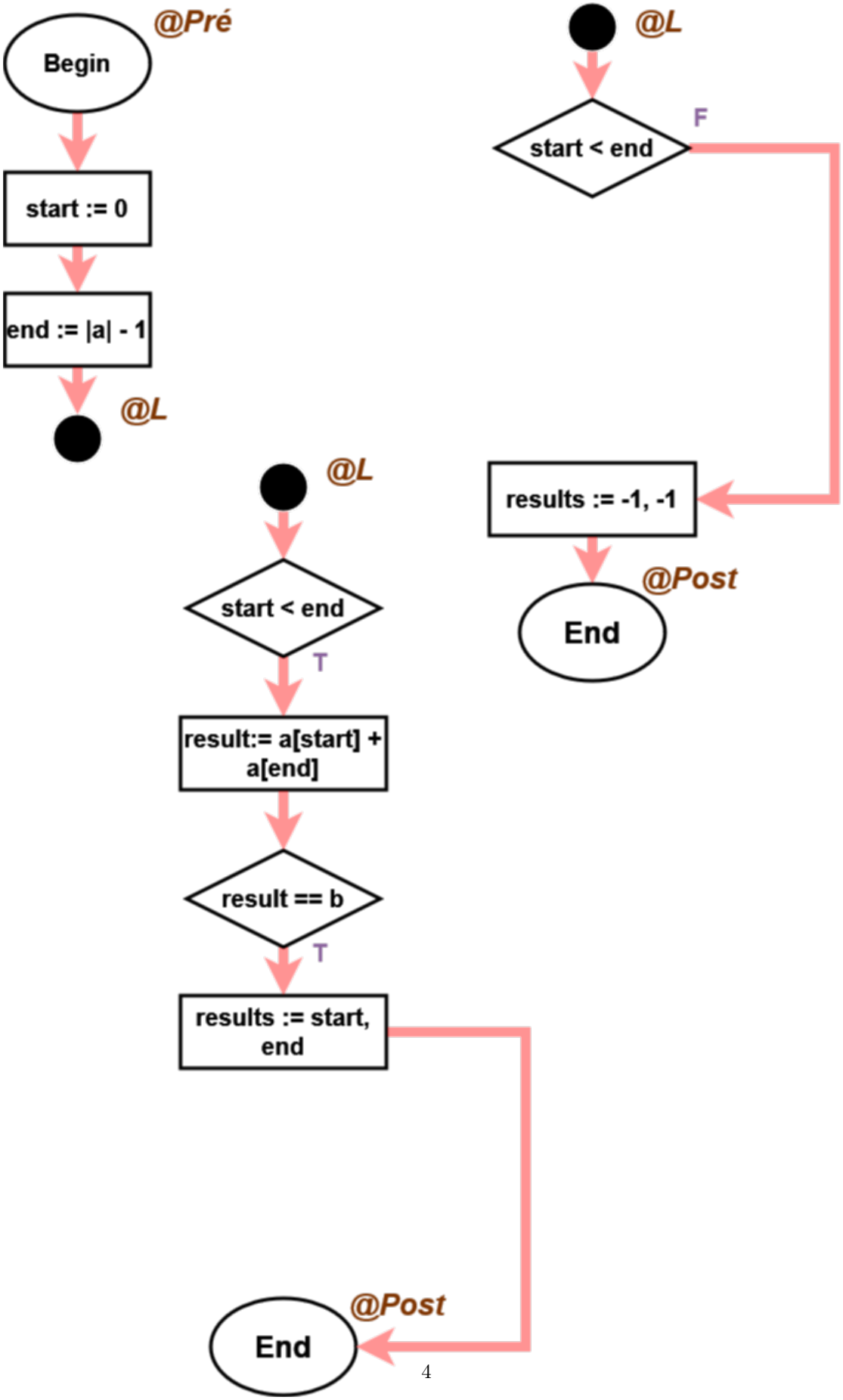
$$end - start$$

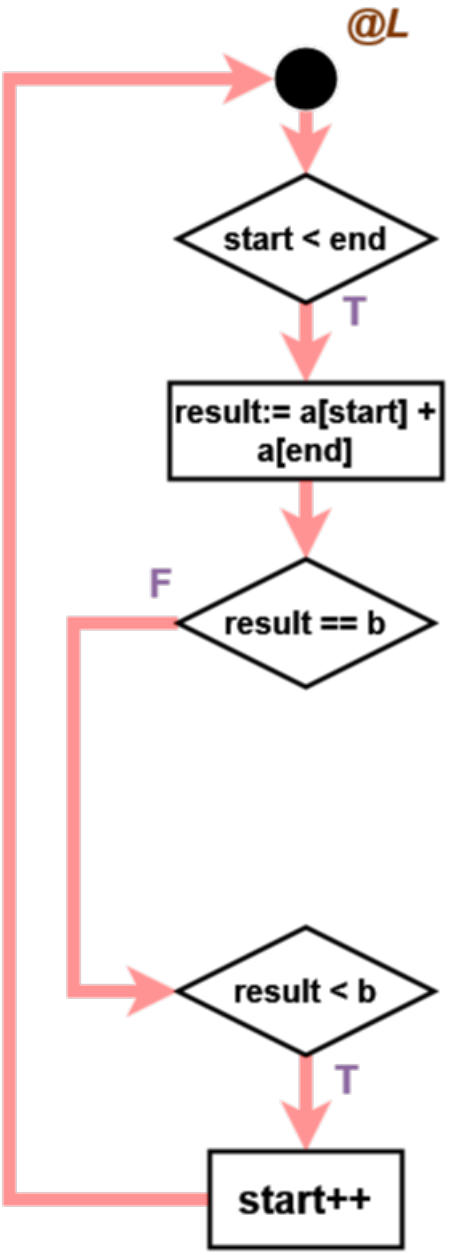
4 Graphe du programme

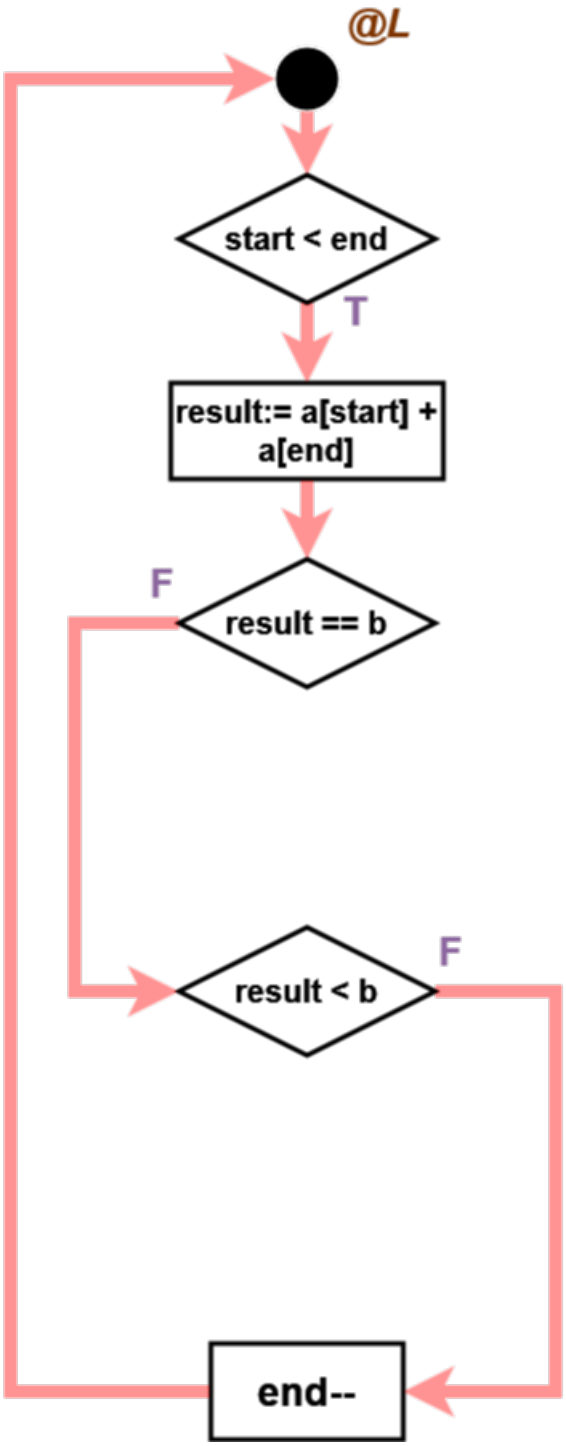
4.1 Graphe complet



4.2 Chemins simples




@L
start < end
T
result:= a[start] + a[end]
F
result == b
result < b
T
start++


@L
start < end
T
result:= a[start] + a[end]
F
result == b
result < b
F
end--

5 Preuves du programme

C.F feuilles scannées

6 Conclusion

Ceci conclu le premier devoir dans le cadre du cours de Méthode de conception de programmes.

Chemin 1

◎ Pré	[(a ≠ null) ∧ sorted (a)]
start := 0	start := 0
end := |a| − 1	end := |a| − 1
◎ L	[∀ i, j ∈ 0 ≤ i < start < j ≤ end => a[i] + a[j] ≠ b]

[∀ i, j ∈ 0 ≤ i < start < j ≤ end => a[i] + a[j] ≠ b]

end := |a| − 1

start := 0

[∀ i, j ∈ 0 ≤ i < 0 < j ≤ |a| − 1 => a[i] + a[j]]

[a ≠ null ∧ sorted (a)]

↳ i commence bien à 0 (me fait penser que cette valeur)

↳ j commence à la fin de la liste → Donc a est non null.

Chemin 2

◎ L	[∀ i, j ∈ 0 ≤ i ≤ start ≤ j ≤ end => a[i] + a[j] ≠ b]
aucune start ≠ end	aucune start ≠ end
results := −1, −1	results := −1, −1
◎ Post	[results := (−1, −1) => ∀ i, j ∈ 0 ≤ i ≤ j < a.length − 1 => a[i] + a[j] ≠ b]

[results := (−1, −1) => ∀ i, j ∈ 0 ≤ i ≤ j < a.length − 1 => a[i] + a[j] ≠ b]

results := −1, −1

[results := −1, −1 => ∀ i, j ∈ 0 ≤ i ≤ j < a.length − 1 => a[i] + a[j] ≠ b]

↳ TRIVIAL

aucune start ≠ end

[results := −1, −1 => ∀ i, j ∈ 0 ≤ i ≤ j < a.length − 1 => a[i] + a[j] ≠ b]

[results := −1, −1 => ∀ i, j ∈ 0 ≤ i ≤ start ≤ j ≤ end => a[i] + a[j] ≠ b]

↳ Trivial OK

Chemin 3

@L

assume start < end

result := a[start] + a[end]

assume result == b

results := start, end

@Post

[∀ i, j ∈ 0 ≤ i ≤ start < j ≤ end => a[i] + a[j] ≠ b]

assume start < end

result := a[start], a[end]

assume result == b

results := start, end

[0 ≤ results[0] ≤ results[1] ≤ a.length - 1 ∧ a[results[0]] + a[results[1]] = b]

[0 ≤ results[0] ≤ results[1] ≤ a.length - 1 ∧ a[results[0]] + a[results[1]] = b]

results := start ; end

[0 ≤ start ≤ end ≤ a.length - 1 ∧ a[start] + a[end] = b]

assume result == b

[0 ≤ start ≤ end ≤ a.length - 1 ∧ a[start] + a[end] = b]

Trivial

result := a[start], a[end]

assume start < end

[0 ≤ start < end => a[i] + a[j] < b]

[∀ i, j ∈ 0 ≤ i ≤ start < j ≤ end => a[i] + a[j] ≠ b]

Chemin 4

@L

omme start < end

result := a[start] + a[end]

omme result ≠ b

omme result < b

start ++

@L

[∀i, j ∈ 0 ≤ i < start < j < end => a[i] + a[j] ≠ b]

omme start < end

result := a[start] + a[end]

omme result ≠ b

omme result < b

start ++

[∀i, j ∈ 0 ≤ i < start < j < end => a[i] + a[j] ≠ b]

[∀i, j ∈ 0 ≤ i < start < j < end => a[i] + a[j] ≠ b]

start ++

omme result < b

omme result ≠ b

result := a[start] + a[end]

omme start < end

[∀i, j ∈ 0 ≤ i < start < j < end => a[i] + a[j] < b]

= result

[∀i, j ∈ 0 < i < start < j ≤ end => result < b]

[∀i, j ∈ 0 ≤ i < start < j ≤ end => a[i] + a[j] ≠ b]

Trivial

Trivial

Chemin S

@L

omme start < end

result := a[start] + a[end]

omme result ≠ b

omme result ≥ b

end --

@L

[∀i, j ∈ 0 ≤ i < start < j ≤ end => a[i] + a[j] ≠ b]

omme start < end

result := a[start], a[end]

omme result ≠ b

omme result ≥ b

end --

[∀i, j ∈ 0 ≤ i < start < j ≤ end => a[i] + a[j] < b]

[∀i, j ∈ 0 ≤ i < start < j ≤ end => a[i] + a[j] ≠ b]

end --

omme result > b

omme result ≠ b

result := a[start], a[end]

omme start < end

[∀i, j ∈ 0 ≤ i < start < j ≤ end => a[i] + a[j] > b]

≡ result

[∀i, j ∈ 0 ≤ i < start < j ≤ end => result > b]

[∀i, j ∈ 0 ≤ i < start < j ≤ end => a[i] + a[j] ≠ b]

Trivial