Étude des tris insertions et sélection

Table des matières

Le problème algorithmique du tri

Nous voulons trier une liste d'élément. Nous pouvons considéré qu'une liste est triée si pour chaque élément dans la liste, celui-ci est inférieur ou égal à son voisin suivant.

Nous pouvons constater dans les exemples ci-dessous que si cette condition est vrai pour chaque élément, le tableau est trié. Si elle est fausse dans un seul cas, le tableau n'est pas trié.

Tableau trié :

2	4	9	11	18

Tableau non trié :

2	4	9	5	18

Nous déduisons les entrées/sorties suivantes :

Entrée : $L \quad \text{tel que} \quad \forall x \in L,\ x \in \mathbb{R}$

Sortie : $\forall x \in [1\,..\,|L|-1], \quad L[x] \leq L[x+1]$

Le tri insertion (ou tri à bulles)

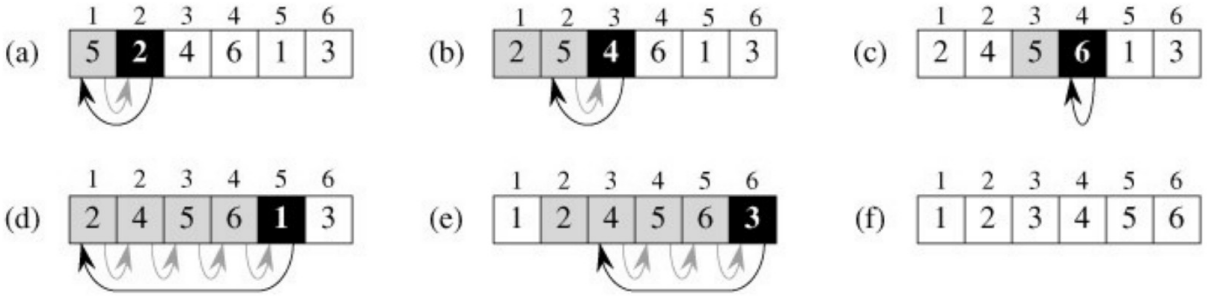
Ce type de tri s'inspire du geste naturel que l'on adopte lorsqu'on trie une main de cartes. Au départ, aucune carte n'est triée : on les prend une par une depuis la zone de cartes non triées, et on les insère à la bonne position dans la zone déjà triée.

Prenons un exemple concret : imaginons qu'un joueur ait déjà trié une partie de ses cartes sur la gauche de sa main — disons (2, 4, 5). Il lui reste encore deux cartes à placer : le 7 et le 10. Il saisit le 7 et le glisse à la bonne place dans la partie triée, ce qui donne (2, 4, 5, 7). Il ne reste alors plus qu'une carte, le 10, à insérer. À chaque étape, la zone triée s'agrandit, tandis que la zone non triée diminue, jusqu'à ce que toutes les cartes soient rangées dans l'ordre.

Explication du code

```
1  Pour i = 2 à |A|
2       clé = A[i]
3       j = i-1
4            Tant que j > 0 et a[j] > clé
5            A[j+1] = A[j]
6       j = j-1
7       A[j+1] = clé
```

La figure ci-dessous illustre le fonctionnement du tri pour une entrée A = [5, 2, 4, 6, 1, 3].



Au cours de l'algorithme, l'indice i pointe la valeur que l'on est en train d'insérer. En début de chaque itération de la boucle pour, la partie du tableau allant de l'indice 1 à i représente les éléments déjà triés, tandis que les éléments de i+1 à |A| restent à trier.

On commence par copier la valeur située à l'indice i+1 dans une variable clé pour la mémoriser, puis on initialise l'indice j pour pointer sur l'élément précédent.

L'objectif de la boucle tant que est alors de trouver la première valeur strictement inférieure à clé dans la partie déjà triée. Comme cette sous-partie est triée en ordre croissant, la parcourir de droite à gauche revient à examiner ses éléments dans l'ordre décroissant. Tant qu'on lit des valeurs supérieures à clé, on continue vers la gauche.

Dès qu'on trouve une valeur inférieure (*ou qu'on atteint le début du tableau*), on sait que toutes les valeurs à droite de j sont supérieures à clé, et celles à gauche ou à j sont inférieures ou égales. On peut donc insérer la clé à la position j+1, qui est exactement sa place dans l'ordre croissant.

La boucle tant que est imbriquée dans la boucle pour. Nous avons d'abords besoin d'étudier le comportement de la boucle tant que afin d'expliquer celui de la boucle pour, l'inverse n'est pas possible.

Étude de la boucle tant que

Variant de boucle

Nous pouvons supposer le variant de boucle suivant :

$$V = j$$

initialisation

Nous cherchons à prouver que le variant est positif avant la première itération de la boucle :

$$j_0 = i - 1 \quad \text{et} \quad 2 \leq i \leq |A|$$

donc

$$1 \leq j_0 \leq i - 1$$

$$j_0 > 0$$

hérédité décroissante

Nous voulons montrer que la suite V_n est de raison négative :

$$j_{n+1} = j_n - 1$$

$$V_{n+1} - V_n = j_{n+1} - j_n = j_n - 1 - j_n = -1$$

terminaison

Démontrons que la boucle s'arrête lorsque $V = 0$:

$$V = 0 \quad \Rightarrow \quad j_t = 0$$

$$j_t = 0 \quad \Rightarrow \quad \text{la condition } (j_t > 0 \wedge \ldots) \text{ est fausse}$$

Invariant de boucle

La boucle tant que part de l'indice i-1 et descend case par case jusqu'à trouver le premier élément du tableau inférieur à la clé. Ni plus, ni moins. Cette boucle à elle seule ne prouve pas le tri du tableau. Elle effectue simplement l'opération décrite précédemment.

Exprimons un variant de boucle :

$$I \equiv \forall x \in \{j + 1 \ldots i\},\ L[x] > \text{clé}$$

Initialisation

Nous voulons prouver que I_0 est vrai. En lisant le code, nous savons :

$$j_0 = i - 1 \quad \text{et} \quad \text{clé} = L[i]$$

donc

$$\begin{aligned} I_0 &\equiv \forall x \in \{i - 1 + 1 \ldots i\},\ L[x] \geq L[i] \\ &\equiv \forall x \in \{i \ldots i\},\ L[x] \geq L[i] \\ &\equiv L[i] \geq L[i] \\ &\equiv \text{vrai} \end{aligned}$$

Hérédité

On veut montrer que si l'invariant est vrai à l'itération I_n il sera vrai à l'itération I_{n+1} .

$$I_n \quad \Rightarrow \quad I_{n+1}$$
$$\forall x \in \{j_n + 1 \dots i\},\ L[x] \geq \text{clé} \quad \Rightarrow \quad \forall x \in \{j_{n+1} + 1 \dots i\},\ L[x] \geq \text{clé}$$

Nous sommes entré dans la boucle, donc nous savons que :

$$L[j_n] \geq \text{clé}$$

En lisant le code nous déduisons :

$$L[j_n + 1] = L[j_n] \Rightarrow L[j_n + 1] \geq \text{clé}$$

Intéressons nous à l'évolution de j. Comme ce dernier augmente, la taille de l'intervalle de l'invariant augmente aussi. Nous démontrons que la seule nouvelle valeur ajoutée à cet intervalle sera j_n.

$$j_{n+1} = j_n - 1, \quad \text{on déduit :} \quad \{j_{n+1} + 1 \dots i\} / \{j_n \dots i\} = \{j_n\}$$

$$\text{On sait que } L[j_n] \geq \text{clé}, \text{ donc couplé à } I_n \text{ nous déduisons :}$$

$$\begin{aligned} \forall x \in \{j_n + 1 \dots i\} \cup \{j_n\},\ L[x] \geq \text{clé} \quad &\equiv \quad \forall x \in \{j_n \dots i\},\ L[x] \geq \text{clé} \\ &\equiv \quad \forall x \in \{j_{n+1} + 1 \dots i\},\ L[x] \geq \text{clé} \\ &\equiv \quad I_{n+1} \end{aligned}$$

En supposant que I_n est vrai, et en déroulant une itération de la boucle supplémentaire, nous avons conclu qu'à la fin de cette boucle $_{n+1}$ nous tombons sur le prédicat I_{n+1} . Nous avons donc démontré que $I_n \Rightarrow I_{n+1}$.

Terminaison

Nous ne pouvons connaître précisément la valeur de j en fin de boucle. Cette dernière s'est potentiellement arrêté avant que j = 0, dans le cas où l'autre partie du et logique a été validée, c'est à dire que nous avons observé un élément inférieur à notre clé.

En sortie boucle nous avons notre invariant et la négation de la condition :

$$\begin{aligned} I_t &\equiv \forall x \in \{j + 1 \dots i\},\ A[x] \geq \text{clé} \wedge \neg(j > 0 \wedge A[j] > \text{clé}) \\ &\equiv \forall x \in \{j + 1 \dots i\},\ A[x] \geq \text{clé} \wedge (j = 0 \vee A[j] \leq \text{clé}) \end{aligned}$$

Étude de la boucle pour

```
1  Pour i = 2 à |A|
2       clé = A[i]
3       j = i-1
4            Tant que j > 0 et a[j] > clé
5            A[j+1] = A[j]
6       j = j-1
7       A[j+1] = clé
```

Variant de boucle

La boucle étudiée est une boucle bornée : par définition, son nombre d'itérations est fini et connu à l'avance. Il n'est donc pas nécessaire d'introduire de variant pour en prouver la terminaison.

La terminaison dépend uniquement d'une question : les instructions imbriquées dans la boucle peuvent-elles bloquer une itération indéfiniment ? Dans notre cas, ces instructions sont élémentaires et se terminent toujours. De plus, nous avons démontré que la boucle tant que imbriquée se termine également. Ainsi, la boucle pour se termine nécessairement.

Invariant de boucle

À chaque itération, la boucle pour prend le premier élément de la partie non triée du tableau et l'insère à la bonne position dans la partie triée. À la fin de chaque itération, la partie triée contient donc un élément supplémentaire, tandis que la partie non triée en contient un de moins.

$$I = \forall (x, y) \in \{1 \dots i-1\}^2,\ x < y \Rightarrow A[x] \leq A[y]$$

On considère plusieurs tableaux illustrant cet invariant :

A	2	3	5	7	9	12
indice	1	2	3	4	5	6

Pour $x = 2$, $y = 5$: $x < y$ et $A[2] = 3 \leq 9 = A[5]$
Pour $x = 1$, $y = 6$: $x < y$ et $A[1] = 2 \leq 12 = A[6]$
Pour $x = 4$, $y = 2$: $x > y$, donc l'implication est vraie par vacuité

A	2	3	8	5	9	12
indice	1	2	3	4	5	6

Pour $x = 1$, $y = 2$: $x < y$ et $A[1] = 2 \leq 3 = A[2]$
Pour $x = 4$, $y = 6$: $x < y$ et $A[4] = 5 \leq 12 = A[6]$
Pour $x = 5$, $y = 3$: $x > y$, donc l'implication est vraie par vacuité

Contre-exemple démontrant que l'invariant est violé :

Pour $x = 3$, $y = 4$: $x < y$ mais $A[3] = 8 > 5 = A[4]$

L'invariant est violé dès qu'un seul couple $(x < y)$ contredit $A[x] \leq A[y]$. En effet le 2e tableau n'est pas trié.

Initialisation

Nous voulons démontrer que l'invariant est vrai à l'itération 0 :

$$I_0 = \forall (x, y) \in \{1 \dots i_0 - 1\}^2,\ x < y \Rightarrow A[x] \leq A[y]$$

Selon le code on connaît la valeur de i_0 :

$$i_0 = 2$$

donc

$$\begin{aligned} I_0 &= \forall (x, y) \in \{1 \dots i_0 - 1\}^2,\ x < y \Rightarrow A[x] \leq A[y] \\ &= \forall (x, y) \in \{1 \dots 1\}^2,\ x < y \Rightarrow A[x] \leq A[y] \\ &= \forall (x, y) \in \{(1, 1)\},\ x < y \Rightarrow A[x] \leq A[y] \\ &= \text{vrai par vacuité (car } x < y \text{ n'est jamais vérifié pour } (1, 1)) \end{aligned}$$

conservation

À l'itération n, nous supposons que l'invariant est vérifié :

$$I_n = \forall (x, y) \in \{1 \dots i_n - 1\}^2,\ x < y \Rightarrow A[x] \leq A[y]$$

Lorsque nous passons à l'itération n+1 :

$$i = n + 1$$

Montrons que l'invariant sera vérifié pour l'itération n+1, nous voulons donc démontrer :

$$I_{n+1} = \forall (x, y) \in \{1 \dots i_{n+1} - 1\}^2,\ x < y \Rightarrow A[x] \leq A[y]$$

Nous allons démontrer cela en 3 cas :

Premier cas : On considère toutes les valeurs sauf celle à la position $j+1$:

$$x, y \in \{1, \ldots, i_{n+1} - 1\} \quad \text{et} \quad x, y \neq j+1$$

L'ordre des éléments considérés n'a pas changé. Ils sont tous : soit à la même place qu'avant, soit décalés d'un cran vers la droite pour laisser une place disponible à l'indice $j+1$. Tous ces éléments restent donc triés entre eux.

Deuxième cas : On compare l'élément introduit en $j+1$ avec les éléments le précédent :

Pour $x \leq j$, les éléments n'ont pas été modifiés et la clé $A[j+1]$ est insérée après tous ceux qui sont inférieurs ou égaux à elle.

Donc pour tout $x \leq j$ on a $A[x] \leq A[j+1]$.

Troisième cas : On compare l'élément introduit en $j+1$ avec les éléments le suivant :

Les éléments d'indice $x > j$ proviennent de la partie triée avant insertion. Lors du décalage, seuls les éléments $A[j+1], A[j+2], \ldots A[i_{n+1}]$ tel que $A[k] > clé$ ont été décalés. Grâce à l'hypothèse de récurrence, nous savons que l'ensemble initial $A[1 \ldots i_{n+1}]$ était trié. Ainsi, tous les éléments déplacés après la position $j+1$ sont supérieurs ou égaux à la clé insérée.

Donc pour tout $y > j+1$ on a $A[j+1] \leq A[y]$.

En combinant nos 3 cas nous concluons que les anciens éléments restent triés entre eux, et que le nouvel élément a été positionné au bon endroit. Nous avons donc démontré que :

$$I_{n+1} = \forall (x, y) \in \{1 \ldots i_{n+1} - 1\}^2,\ x < y \Rightarrow A[x] \leq A[y]$$

terminaison

Lorsque la boucle pour se termine, la variable i atteint la valeur $|A| + 1$. Ainsi, à la sortie de la boucle on a :

$$i = |A| + 1$$

on remplace i par sa valeur $|A| + 1$, ce qui donne :

$$I_{\text{final}} = \forall (x, y) \in \{1 \ldots |A|\}^2,\ x < y \Rightarrow A[x] \leq A[y]$$

Autrement dit, le tableau A est trié en ordre croissant.

Complexité temporelle

L'algorithme contient :

- Une boucle principale pour de $i = 2$ à $|A|$, donc $|A| - 1$ itérations.
- À chaque itération, une boucle tant que qui peut déplacer plusieurs éléments.

À l'itération i, on effectue jusqu'à $i - 1$ comparaisons et décalages.

Par exemple :

- À $i = 2$, au plus 1 comparaison,
- À $i = 3$, au plus 2 comparaisons,
- À $i = 4$, au plus 3 comparaisons,
- etc.

Donc, le nombre total d'opérations effectuées dans l'algorithme est :

$$1 + 2 + 3 + \ldots + (|A| - 1)$$

Ce qui s'écrit :

$$\sum_{i=1}^{|A|-1} i$$

Ce qui vaut :

$$\frac{n(n+1)}{2}$$

En remplaçant n par $|A| - 1$, nous obtenons :

$$\frac{(|A| - 1)|A|}{2}$$

En conclusion, Le temps d'exécution dans le pire cas est $\mathcal{O}(|A|^2)$.