Rapport du projet d'Algorithmique INFOB237

Esteban Bernagou
Martin Devolder
Virgile Devolder

Mai 2023

1 Exercice 1

1.1 Analyse de l'énoncé

Le 2*n lignes du fichier input est ambigu dans la consigne. Il ne nous sert pas à résoudre le problème.

1.2 Specs JML

/*
@requires n > 0; // le nombre de rangées est positif
@requires \nonnullelements(rows); // les rangées ne sont pas nulles
@requires (\forall int i; 0 ≤ i ∧ i < rows.length; rows[i] > 0); // le nombre de plantes est strictement positif pour chaque rangée
@ensures \result == null || (\exists int i; 0 ≤ i ∧ i < rows.length;\result.equals(invasive(rows[i])));
// la fonction renvoie null ou le nom d'une plante envahissante pour chaque rangée
@*/

1.3 Algorithme naïf

On utilise une liste pour stocker les fleurs ainsi qu'une Hashmap pour stocker le nombre d'occurences de ces fleurs dans notre liste.
Voir code pour plus de détails

1.4 Correction d'invariant et terminaison de clôture

1.4.1 Première boucle

$P \equiv a = a0 \land N = \text{listFlower.size()} \land \text{counts vide}$
$I \equiv a = a0 \land \text{counts} \mathrel{+}= \text{plant.occurences()} \mid \text{plant} \in \text{listFlowers}$

Correction de l'invariant :

Au début counts est vide à chaque itération. A chaque itération on rajoute le nombre d'occurences de la plante courante dans counts. A la fin counts contient le nombre d'occurences de plant dans listFlowers.
En vu de tout cela on obtient bien :
$P \Rightarrow I \wedge sp(S, I \wedge B) \Rightarrow I$
S : Instrction du corps de la boucle
B : Condition de terminaison de boucle

<u>Terminaison de clôture :</u>

$V = N \mid N$: nombre d'éléments restants à parcourir dans listFlowers
on a :
$sp(S, I \wedge B \wedge V = n) \Rightarrow V < n \mid n$: taille de listFlowers

Terminaison si :
$V = 0 \Rightarrow \neg B$
or V = V - 1 à chaque itération,
On obtient finalement : $V = 0 \equiv N = 0$

1.4.2 Deuxième boucle

$P \equiv a = a0 \wedge N =$ counts.keyset().size() $\wedge$ counts non vide
$I \equiv a = a0 \wedge$ plant.occurences() $<$ listFlowers.size() / 2 $\mid$ plant $\in$ listFlowers

<u>Correction de l'invariant :</u>

Au début de la boucle, tooMuch est null. Si le nombre d'occurrences de l'élément courant dans la table de hachage counts est supérieur à la moitié de la taille de la liste listFlowers, alors tooMuch prend la valeur de l'élément courant. À la fin de la boucle, tooMuch contient l'élément qui apparaît plus de la moitié du temps dans la liste listFlowers, s'il en existe un.
En vu de tout cela on obtient bien :
$P \Rightarrow I \wedge sp(S, I \wedge B) \Rightarrow I$
S : Instrction du corps de la boucle
B : Condition de terminaison de boucle

<u>Terminaison de clôture :</u>

$V = N \mid N$: nombre d'éléments à parcourir dans counts on a :
$sp(S, I \wedge B \wedge V = n) \Rightarrow V < n \mid n$: taille de counts

Terminaison si :
$V = 0 \Rightarrow \neg B$
or V = V - 1 à chaque itération,
On obtient finalement : $V = 0 \equiv N = 0$

1.5 Complexité de l'algorithme naïf

1.5.1 Première boucle

Complexité O(n) car on itère une fois sur chaque élément

1.5.2 Deuxième boucle

Complexité O(n) car on itère une fois sur chaque élément

1.5.3 Conclusion

O(n + n) = O(2n) = O(n) ⇒ complexité linéaire

1.6 Solution plus efficace

Cette fois nous avons choisi d'implémenter un algorithme récursif.
Voir code pour plus de détails

1.7 Specs JML algorithme efficace

Voir code

1.8 Solution "diviser pour mieux régner

Voir code

1.9 Complexité de "Diviser pour mieux régner"

Utilisons la formule récuuente : $T(n) = c * T(n/d) + b * n^k$
Nombre de sous-problèmes à chaque récursion : 2
Taille de chaque sous problème : $n/2$

Donc $T(n) = 2 * T(n/2) + O(n)$ avec O(n) le coût de recherche du nombre d'occurences d'une plante dans une sous-liste

<u>Conclusion :</u>
k = 1, c = 2 et d = 2 et $c = d^k \iff 2 = 2^1$
Complexité de l'algorithme : $O(n^k * log(n) = O(n * log(n)$
⇒ Complexité linéarithmique