Algorithmique
Partie 5 – Invariants de boucle

Première NSI

Lycée Vaugelas, Chambéry

Ce cours a pour but d'introduire du vocabulaire autour de la notion d'exactitude d'un algorithme.

Puis d'aborder la notion de preuves d'algorithme.

Et pour finir, on abordera une méthode utilisée pour prouver l'exactitude d'un algorithme : les invariants de boucle.

Plan du cours

Plan du cours

Validation d'un programme

Exactitude – Preuves d'algorithme

Invariant de boucle
- Un peu de logique
- Exemple d'invariant de boucle

Que signifie valider un programme ?

→ c'est montrer qu'il correspond à ses spécifications

Rappel : spécifications d'un programme

Il y a 2 types de spécifications d'un programme :

- les pré-conditions
- les post-conditions

Qu'est-ce que l'exactitude d'un algorithme ?
Quand dit-on qu'un algorithme est exact ?

1) il faut montrer que l'algorithme termine
2) il faut montrer qu'une fois terminé, l'algorithme a fourni la réponse exacte

Plan du cours

Définition

On parle de terminaison d'un programme quand l'exécution du programme produit un résultat en un temps fini, quelques soient les données en entrées

Donner la preuve de terminaison d'un programme ne dit rien sur la qualité du résultat fourni. Ce qui nous amène au deuxième type de preuve :

Définition

Une fois l'exécution du programme terminée, il y a correction partielle de l'algorithme si le résultat fourni en sortie est le résultat exact quelque soient les données fournies en entrée.

Ainsi, prouver l'**exactitude** d'un algorithme, ou faire la **correction totale**, c'est donner

- la preuve de **terminaison**
- la preuve de **correction partielle**

Plan du cours

Une méthode répandue pour prouver l'exactitude d'un programme est de travailler sur un **invariant de boucle**.

Pour qu'un invariant de boucle nous aide à vérifier l'exactitude d'un algorithme, nous devons montrer les 3 points suivants :

- **initialisation** : l'invariant est vrai avant la 1ère itération de la boucle
- **conservation** : si l'invariant est vrai avant une itération de la boucle, il le reste avant l'itération suivante
- **terminaison** : la boucle se termine : l'invariant de boucle, ainsi que la raison de la fin de boucle, nous fournissent une information utile

Plan du cours

On considère un raisonnement du type :

Si la proposition A est vraie alors la proposition B est vraie

Ce raisonnement est représenté par la formulation mathématique suivante

$$A \Rightarrow B$$

La notation mathématique $\bar{B}$ signifie contraire de la proposition B

On appelle contraposée de $A \Rightarrow B$ le raisonnement $\bar{B} \Rightarrow \bar{A}$.

Le point important est que

- le raisonnement $A \Rightarrow B$
- et sa contraposée $\bar{B} \Rightarrow \bar{A}$

sont équivalents d'un point de vue mathématique. Si l'un est vrai, l'autre l'est aussi.

Plan du cours

On reprend l'algorithme 2 : `meilleure_recherche_linéaire`

Algorithme 2 : meilleure_recherche_linéaire (A, n, x)
Entrées :
- A : tableau non trié
- n : nb d'éléments dans A
- x : valeur recherchée dans A

Sorties :
- soit indice i tel que $A[i] = x$
- soit valeur spéciale « Non Trouvé »

Procédure :
1. Pour $i = 1$ à n,
   A. Si $A[i] = x$, alors retourner i
2. Retourner la valeur « Non Trouvé »

Algorithme 2 : meilleure_recherche_linéaire (A, n, x)
Procédure :
1. Pour i = 1 à n,
   A. Si A[i] = x, alors retourner i
2. Retourner la valeur « Non Trouvé »

Comment prouver l'exactitude de cet algorithme ?

Etape 1 : la terminaison du programme est évidente du fait de la boucle de l'étape 1 : `Pour i = 1 à n` compris
Dès que `i>n`, la boucle s'arrête.

Algorithme 2 : meilleure_recherche_linéaire (A, n, x)
Procédure :
1. Pour i = 1 à n,
   A. Si A[i] = x, alors retourner i
2. Retourner la valeur « Non Trouvé »

Etape 2 : est-ce que le résultat renvoyé est exact ?
Il y a deux réponses possibles :
- soit la réponse de l'étape 1A → étape 2a
- soit la réponse de l'étape 2 → étape 2b

Etape 2a : si la procédure renvoie autre chose que « Non Trouvé » alors l'indice renvoyé est correct puisque la seule raison pour que la procédure retourne un indice à l'étape 1A est que `x = A[i]`.

Algorithme 2 : meilleure_recherche_linéaire (A, n, x)
Procédure :
1. Pour i = 1 à n,
   A. Si A[i] = x, alors retourner i
2. Retourner la valeur « Non Trouvé »

Etape 2b : on va utiliser l'invariant de boucle pour montrer que si la procédure retourne la valeur « Non Trouvé » alors x n'est pas dans le tableau A.

Invariant de boucle

A l'itération i de la boucle,
si x est présent dans le tableau A,
alors x est présent dans le sous-tableau `A[i..n]`

Invariant de boucle

A l'itération i de la boucle,
si x est présent dans le tableau A,
alors x est présent dans le sous-tableau `A[i..n]`

invariant de boucle → initialisation / conservation / terminaison

Etape 2b : initialisation `i=1`
Le sous-tableau de l'invariant est `A[1..n]`, soit le tableau A entier.
Si x est dans le tableau A, alors x est dans le sous-tableau `A[1..n]`.
L'initialisation de l'invariant est correcte.

Algorithme 2 : meilleure_recherche_linéaire (A, n, x)
Procédure :
1. Pour i = 1 à n,
   A. Si A[i] = x, alors retourner i
2. Retourner la valeur « Non Trouvé »

Etape 2b : conservation
Je suppose que l'invariant de boucle est vrai au début de la boucle à l'itération i. Donc je suppose que si x est dans le tableau A, alors il est dans le sous-tableau `A[i..n]`.
On parcourt la boucle qui ne contient qu'une seule instruction. On regarde ici pourquoi la procédure renvoie « Non Trouvé » donc si l'étape 1A ne renvoie rien, c'est que $A[i] \neq x$.

Algorithme 2 : meilleure_recherche_linéaire (A, n, x)
Procédure :
1. Pour i = 1 à n,
   A. Si A[i] = x, alors retourner i
2. Retourner la valeur « Non Trouvé »

Etape 2b : conservation
Je suppose que l'invariant de boucle est vrai au début de la boucle à l'itération i. Donc je suppose que si x est dans le tableau A, alors il est dans le sous-tableau `A[i..n]`.
On parcourt la boucle qui ne contient qu'une seule instruction. On regarde ici pourquoi la procédure renvoie « Non Trouvé » donc si l'étape 1A ne renvoie rien, c'est que $A[i] \neq x$.
Donc à la fin de la boucle, on sait que si x est dans le tableau A, il est dans le sous-tableau `A[i..n]`. Et on sait que $A[i] \neq x$, donc si x est dans A, x est dans le sous-tableau `A[i+1..n]`. Comme i est incrémenté avant l'itération suivante (étape 1), l'invariant de boucle est conservé avant l'itération suivante.

Algorithme 2 : meilleure_recherche_linéaire (A, n, x)
Procédure :
1. Pour i = 1 à n,
   A. Si A[i] = x, alors retourner i
2. Retourner la valeur « Non Trouvé »

Etape 2b : terminaison
Cette boucle se termine, soit parce que la procédure renvoie i à l'étape 1A (cas déjà traité), soit parce que `i>n`.

Le plus simple ici est de montrer la contraposée de l'invariant de boucle : si x n'est pas dans le sous-tableau `A[i..n]`, alors x n'est pas dans A.

Comme `i>n`, le sous-tableau `A[i..n]` est vide et ne peut donc contenir x, donc x n'est pas dans A.
La contraposée est prouvée, elle est équivalente à l'invariant de boucle qui est donc prouvé.

Exercices

Fiche d'exercice `05_algo_eleve.pdf` sur les propositions logiques, contraposée, ...

A votre programme, vous avez les invariants de boucle des tri par insertion et tri sélection. Voir les deux exercices sur le site, dans la rubrique `Algorithmique / Preuve d'algorithme et Invariant de boucle` $\rightarrow$ Exercices 1 et 2.
Pour aborder cette question d'invariant de boucle, il faut bien revoir ce que fait chaque algorithme de tri.