Université Clermont Auvergne
ISIMA 1A par apprentissage
Algorithmique

Florent Foucaud
2022-2023

TD1 - Terminaison et correction

1 Rappel de cours

1.1 Terminaison d'un algorithme

Pour montrer qu'un algorithme se termine, il faut montrer qu'on va vérifier une condition de terminaison. Par exemple, qu'on va sortir de la boucle principale, ou bien, qu'on ne fait pas une infinité d'appels récursifs.

Pour une boucle, on peut par exemple montrer que la suite des valeurs de la variable principale (celle du test de terminaison de la boucle) converge vers la valeur de terminaison.

Pour un algorithme récursif, on peut montrer que la suite des valeurs des appels récursifs converge vers le cas de base de la fonction récursive.

1.2 Correction d'un algorithme

Il s'agit de montrer que l'algorithme est correct, c'est à dire qu'il calcule bien le bon résultat.

Pour montrer qu'un algorithme utilisant une boucle principale est correct, on peut utiliser un *invariant de boucle* : en notant par j le nombre d'itérations de la boucle, c'est une propriété $P(j)$ qui est vraie avant la boucle ($j=0$) ou bien après le premier tour de boucle ($j=1$), qui sera vraie après chaque itération de la boucle (on passe de $P(j)$ à $P(j+1)$), et qui, pour la dernière itération, montre que le résultat calculé par l'algorithme est correct. On montre le passage de $P(j)$ à $P(j+1)$ généralement par récurrence sur le nombre d'itérations de la boucle.

Pour un algorithme récursif, on utilise naturellement une preuve par récurrence sur les appels récursifs.

2 Exercices

Exercice 1 (Terminaisons)**.**

Les algorithmes suivants se terminent-ils ? Pourquoi ? Si non, dans quels cas particuliers se terminent-ils ?

```
Algo1(n)
Entrée : un entier n
Sortie : un entier
  • Tant que n≠0 faire :
      ⋆ Si n est pair alors :
        n = 2n
      ⋆ Sinon :
        n = n−1
  • Retourner n
```

```
Algo2(n)
Entrée : un entier n
Sortie : un entier
  • Tant que n > 1 faire :
      ⋆ Si n est pair alors :
        n = n/2
      ⋆ Sinon :
        n = n+1
  • Retourner n
```

```
Algo3(n)
Entrée : un entier n
Sortie : un entier
  • Retourner n + Algo3(n−1)
```

```
Algo4(n)
Entrée : un entier n
Sortie : un entier
  • Si n==10 alors :
    Retourner 1
  • Sinon :
    Retourner n − Algo4(n−1)
```

Exercice 2 (Maximum itératif)**.**

Montrer que l'algorithme suivant se termine. Montrer qu'il est correct en utilisant un invariant de boucle.

`maxiter(`T`): Recherche du maximum d'un tableau` T

`Entrée : un tableau` $T[1...n]$ `non trié de` n `entiers`

`Sortie : le plus grand entier de` T

- `max` $=$ $T[1]$
- `Pour` i `allant de` 2 `à` n `:`
  - `Si` $T[i] > $ `max alors :`
    `max` $=$ $T[i]$
- `Retourner max`

Exercice 3 (Maximum récursif)**.**

Montrer que l'algorithme suivant se termine. Montrer qu'il est correct en utilisant un raisonnement par récurrence sur la taille du tableau.

`maxrec(`T`): Recherche du maximum d'un tableau` T

`Entrée : un tableau` $T[1...n]$ `non trié de` n `entiers`

`Sortie : le plus grand entier de` T

- `Si` $n==1$ `alors :`
  `Retourner` $T[1]$
- `Sinon :`
  `Retourner le maximum entre maxrec(`$T[1...n-1]$`) et` $T[n]$

Exercice 4 (Tri par sélection)**.**

Dans le tri par sélection, on trie de gauche à droite et on choisit le plus petit élément dans la partie non triée et on le met en fin de la partie triée. Montrer que l'algorithme se termine. Montrer qu'il est correct en utiliant des invariants de boucle. Il faudra raisonner séparément sur les deux boucles imbriquées.

`triSelection(`T`): tri d'un tableau` T `d'entiers par ordre croissant`

`Entrée : un tableau` $T[1...n]$ `non trié de` n `entiers`

`Sortie : le tableau` T `trié`

- `Pour` i `allant de` 1 `à` $n-1$ `faire :`
  - ⋆ `indiceMin` $= i$
  - ⋆ `valeurMin` $= T[i]$
  - ⋆ `Pour` j `allant de` $i+1$ `à` n `faire :`
    - − `Si` $T[j] <$ `valeurMin alors :`
      `indiceMin` $=$ `j`
      `valeurMin` $= T[j]$
  - ⋆ T[`indiceMin`] $= T[i]$
  - ⋆ $T[i] =$ `valeurMin`
- `Retourner` T

Exercice 5 (Tri par insertion)**.**

Dans l'algorithme de tri par insertion, on trie le tableau de gauche à droite, en insérant la valeur courante à la bonne place dans la partie déjà triée. Montrer que l'algorithme se termine. Montrer qu'il est correct en utiliant un invariant de boucle. Il faudra raisonner séparément sur les deux boucles imbriquées.

`triInsertion(`T`): tri d'un tableau` T `d'entiers par ordre croissant`

`Entrée : un tableau` $T[1...n]$ `non trié de` n `entiers`

`Sortie : le tableau` T `trié`

- `Pour` i `allant de` 2 `à` n `faire :`
  - ⋆ $j = i-1$
  - ⋆ `valeurInsertion` $= T[i]$
  - ⋆ `Tant que` $j > 0$ `et` $T[j] >$ `valeurInsertion faire :`
    - − $T[j+1] = T[j]$
    - − $j = j-1$
  - ⋆ $T[j+1] =$ `valeurInsertion`
- `Retourner` T

Exercice 6 (Recherche dichotomique récursive)**.**

Montrer que l'algorithme suivant se termine. Montrer qu'il est correct par un raisonnement par récurrence sur $n-m$.

`rechercheDicho(`T`,`m`,`n`,`x`): calcul d'une position de` x `dans` T

`Entrée : Un tableau d'entiers` $T[m...n]$ `triés par ordre croissant, un entier` x`, deux positions` m `et` n

`Sortie :` 0 `si` $x \notin T$`, et une position de` x `dans` T `(entre` m `et` n`) sinon`

- `Si` $m == n$ `:`
  - ⋆ `Si` $T[m] == x$`:`
    `Retourner` m
  - ⋆ `Sinon :`
    `Retourner` 0
- `Sinon :`
  - ⋆ $k = \lfloor \frac{m+n}{2} \rfloor$
  - ⋆ `Si` $T[k] < x$ `:`
    `Retourner rechercheDicho(`T`,`$k+1$`,`n`,`x`)`
  - ⋆ `Sinon :`
    `Retourner rechercheDicho(`T`,`m`,`k`,`x`)`

Exercice 7 (Couplage stable – Gale-Shapley, 1962)**.**

Étant donnés deux ensembles A et B, un *couplage* est un ensemble de paires de $A \times B$, deux à deux disjointes. On applique cette notion par exemple pour apparier des étudiants à des places en formation, des donneurs d'organes à des receveurs, etc. On représente un couplage par une application $c : A \to B$ qui associe à chaque élément a de A un élément $c(a)$ de B. Si $c(a) = b$, on notera $a = c^{-1}(b)$.

Lorsque les éléments de A et B ont des listes de préférence, on parle de *couplage stable* $c : A \to B$ si pour toute paire (a,b) de $A \times B$ non-appariée, a ne préfère pas b par rapport à son élément apparié $c(a)$ et b ne préfère pas a à son élément apparié $c^{-1}(b)$.

Gale et Shapley ont démontré qu'il existe toujours un couplage stable, grâce à l'algorithme ci-dessous qui permet d'en produire un.

Montrer que l'algorithme termine, et qu'il est correct.

`couplageStable(`A`,`B`)`: calcul d'un couplage stable entre A et B

Entrée : Deux ensembles A et B et pour chaque élément x, une liste ordonnée de préférences $L(x)$ d'éléments de l'autre ensemble

Sortie : Un couplage stable de A,B

- Tant qu'il existe un élément de A non apparié avec une liste de préférences non vide :
  - ⋆ Choisir un tel élément a de A
  - ⋆ Soit b l'élément de $L(a)$ préféré par a
  - ⋆ Si b n'est pas apparié :
    Apparier a et b
  - ⋆ Sinon (b est apparié à $c^{-1}(b)$), si b préfère a à $c^{-1}(b)$:
    Dés-apparier b et $c^{-1}(b)$
    Apparier a et b
  - ⋆ Sinon :
    Enlever b de $L(a)$
- Retourner le couplage