TD 2 : Correction et terminaison d'algorithmes

12 septembre 2024

Page web du cours : `https://people.irisa.fr/David.Baelde/algo/index.html`
Coordonnées des chargés de ce TD : `malo.revel@ens-rennes.fr` `pablo.espana-gutierrez@inria.fr`
Les TDs ont lieu les jeudis de 9h45 à 11h15, se fier à ADE pour la salle.

1 Mariages stables

Rappel de l'algorithme de Gale-Shapley :
(On note $P(a) \in B$ le $(Index[a])^{\text{ème}}$ choix de a.)

```
1  C := ∅ ; Index := [0; ...; 0]
2  tant que C ≠ A faire
3      choisir a ∈ A \ C
4      b := P(a)
5      si ∃a' ∈ C. b = P(a') alors
6          si b préfère a à a' alors
7              C := C ∪ {a} \ {a'}
8              Index[a'] ← 1 + Index[a']
9          sinon
10             Index[a] ← 1 + Index[a]
11     sinon
12         C := C ∪ {a}
```

1.1 Déroulement sur un exemple

La figure 1 présente quatre hôpitaux H1, H2, H3, H4 et quatre accidents A1, A2, A3, A4. On cherche à affecter exactement un accident à chaque service d'urgence. Chaque hôpital (resp. accident) a un ordre de préférence basé sur sa distance aux accidents (resp. hôpitaux).

Question 1 – Appariez ceux-ci en utilisant l'algorithme de Gale-Shapley, en parcourant les accidents par indice croissant.

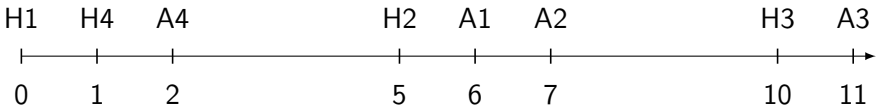

FIGURE 1 – Hôpitaux et accidents

1.2 Correction de l'algorithme

On se concentre ici seulement sur une partie de la preuve de l'invariant proposé en cours de l'algorithme de Gale-Shapley. Pour rappel, l'invariant est la conjonction des quatre propriétés suivantes :

P_1 : $C \subseteq A$

P_2 : $\forall a \in A.\ Index[a] \in [\![0, n-1]\!]$

P_3 : $\forall a \neq a' \in C.\ P(a) \neq P(a')$

P_4 : $\forall a \in A, b \in B$ tels que a préfère b à $P(a)$,
$\exists a' \in C$ tel que $b = P(a')$ et b préfère a' à a

Supposons que P_1, P_2, P_3 et P_4 sont vrais au début d'une exécution du corps de la boucle. On souhaite alors montrer que P_2 est encore vrai à la fin de cette exécution.

Notons C' et $Index'$ l'état respectivement de C et de $Index$ à la fin de cette exécution.

QUESTION 2 – Montrer que si $Index[a] \geq n-1$, alors $Index'[a] \in [\![0, n-1]\!]$.

QUESTION 3 – Montrer que s'il existe $a' \in C$ tel que $b = P(a')$, alors $Index[a'] < n-1$.

QUESTION 4 – En déduire que P_2 est vrai à la fin de l'exécution du corps de boucle.

2 Correction d'algorithmes

2.1 Mélange de Fisher-Yates

On suppose disposer d'une fonction permettant de générer des entiers tirés uniformément au hasard de manière indépendante dans un intervalle donné. On cherche à permuter les éléments d'un tableau T de n éléments supposés différents en suivant une permutation tirée uniformément au hasard parmi toutes les permutations à n éléments possibles.

Précondition: T un tableau non vide
Fonction FISHER-YATES(T)
  $i \leftarrow 0$
  pour i allant de 1 à $|T| - 1$ inclus **faire**
    $k \leftarrow$ entier tiré au hasard entre 0 et i inclus
    $T[k] \leftrightarrow T[i]$
  return T

QUESTION 5 – Proposer un invariant de boucle qui permettra de montrer ce point.

QUESTION 6 – **(Bonus)** Montrer que cet invariant est bien respecté initialement, et qu'il est bien conservé après chaque tour de boucle.

QUESTION 7 – **(Bonus)** L'algorithme reste-t-il correct en tirant k entre 0 et $i-1$ inclus à la place ?

2.2 Suppression du dernier élément d'une liste chaînée (Bonus)

Pour prouver la correction d'un algorithme récursif, on n'utilise pas d'invariant de boucle mais une propriété prouvée récursivement (plus généralement par induction) sur les arguments de la fonction.

Précondition: $l \neq []$
  Fonction $\mathrm{POP}(l)$
    si $l = [x]$ **alors**
      return $[]$
    si $l = h :: t$ **alors**
      return $h :: (\mathrm{POP}(t))$

QUESTION 8 – Montrer que cet algorithme renvoie la liste non vide donnée en argument, privée de son dernier élément.

2.3 Fonction 91 de McCarthy (Récursive)

La fonction suivante, appelée la fonction 91 de McCarthy a une propriété étonnante : malgré son expression sophistiquée, pour tout $n \leq 100$ elle renvoie 91 (pour $n > 100$, elle renvoie $n - 10$).

  Fonction $\mathrm{M91}(n)$
    si $n > 100$ **alors**
      return $n - 10$;
    sinon
      return $\mathrm{M91}(\mathrm{M91}(n + 11))$

QUESTION 9 – Montrer que pour tout $n \leq 100$, M91(n) renvoie 91.

3 Terminaison d'algorithmes

3.1 Définitions

Symétriquement à la notion d'invariant utilisée pour la preuve de correction d'algorithme, une notion de variant peut être utilisée pour établir leur terminaison. On rappelle ici quelques définitions utiles.

Définition 1 (Ordre bien fondé)**.** *Soit $(E, \preccurlyeq)$ un ensemble ordonné. On dit que l'ordre $\preccurlyeq$ est bien fondé s'il n'existe pas de suite infinie strictement décroissante d'éléments de E.*

Exemple 1. *$(\mathbb{N}, \leq)$ est un ensemble bien fondé, mais $(\mathbb{Z}, \leq)$ n'est pas un ensemble bien fondé.*

On peut alors exploiter cette propriété pour garantir qu'une boucle ou une fonction récursive ne pourra itérer qu'un nombre fini de fois.

Définition 2 (Variant). *Soit $(E, \preccurlyeq)$ un ensemble bien fondé.*

Cas itératif : *un variant de boucle est une fonction V à valeur dans E, qui dépend des variables du programme (l'état courant), et telle que si s est un état valide (satisfaisant l'invariant de boucle) en début de boucle, et s' l'état résultant après exécution d'un tour de boucle, alors on a $V(s') \prec V(s)$.*

Cas récursif : *un variant récursif est une fonction V : $\texttt{Arg} \to E$ telle que si arg est la valeur des arguments lors de l'appel de la fonction, et arg' la valeur des arguments lors d'un de ses appels récursifs, alors on a $V(arg') \prec V(arg)$.*

L'existence d'un variant est une condition suffisante à la terminaison d'un programme.

Définition 3 (Ordre lexicographique). *Soit $(E, \preccurlyeq_E)$ et $(F, \preccurlyeq_F)$ deux ensembles bien fondés. On définit l'ordre lexicographique sur leur produit cartésien $(E \times F, \preccurlyeq_{lex})$ par :*

— *si $e_1 \prec_E e_2$, alors pour tout f_1, f_2, $(e_1, f_1) \preccurlyeq_{lex} (e_2, f_2)$;*

— *si $f_1 \preccurlyeq_F f_2$, alors pour tout e, $(e, f_1) \preccurlyeq_{lex} (e, f_2)$*

QUESTION 10 – Montrer que si $(E, \preccurlyeq_E)$ et $(F, \preccurlyeq_F)$ sont bien fondés, alors $(E \times F, \preccurlyeq_{lex})$ est bien fondé.

QUESTION 11 – Pourquoi s'intéresser à des variants à valeur dans des ensembles bien fondés plus sophistiqués que $(\mathbb{N}, \leq)$? Autrement dit, pourquoi ne pas utiliser que des variants à valeur entière ?

3.2 Fonction 91 de McCarthy (impérative)

Fonction $\mathrm{M91I}(n)$
  $s \leftarrow 1$
  tant que $s > 0$ **faire**
    si $n > 100$ **alors**
      $n \leftarrow n - 10$
      $s \leftarrow s - 1$
    sinon
      $n \leftarrow n + 11$
      $s \leftarrow s + 1$
  renvoyer(n)

QUESTION 12 – Justifier rapidement (sans invariant) pourquoi cet algorithme se comporte bien comme la fonction 91 de McCarthy vue précédemment.

QUESTION 13 – Montrer que cet algorithme termine à l'aide d'un variant de boucle.

Indice : on peut s'intéresser pour le variant de boucle à la fonction

$$V(s, n) = (101 - n + 10 * s, s)$$

3.3 Somme d'entiers fastidieuse (Bonus)

Précondition: Tous les éléments de l sont ≥ 0
Fonction $\mathrm{SOMME}(l)$
si $l = []$ **alors**
return 0
si $l = h :: t$ **alors**
si $h = 0$ **alors**
return $\mathrm{SOMME}(t)$
sinon
return $1+ \mathrm{SOMME}((h-1) :: t)$

Question 14 – Prouver la terminaison de la fonction SOMME à l'aide d'un variant de boucle.

À rédiger pour la semaine prochaine

Au choix parmi 3 options :
— Questions 5, 6, 7 et 8 (invariants)
— Questions 9, 12 et 13 (McCarthy)
— Questions 10 et 14 (variants)