Université Toulouse III Paul Sabatier

Université de Toulouse

Faculté des Sciences et d'Ingénierie

Terminaison : CTD4

Terminaison/Variant

Algorithmique

Semestre 2 - 2019-2020

DM1

- sur Pixal et Moodle. Commencer **par moodle** pour avoir le code du groupe pixal "DM1 Algo 19-20".
- à rendre au plus tard lundi 24/02 à 12h.
- sur pixal utiliser uniquement le login pixal au format "ups_nomp123" pour authentification.
- sur pixal, les restrictions d'invariants sont à respecter impérativement : contenu et ordre des asserts, boucle while, fonction...

Rappel

Etats, variant et terminaison

- Etats
- Variant

Méthologie complète : exercices

- Objectif et Terminaison
- Pivot
- Drapeau Hollandais

Rappel

Etats, variant et terminaison

Méthologie complète : exercices

Pour formaliser la résolution d'un algorithme de répétition :

- a) identifier et préciser l'objectif
- b) selon un modèle de solution obtenir à partir de l'objectif : l'Invariant et la condition d'arrêt
- c) vérifier la terminaison

Une fois a) et b) définis, le schéma de résolution est le suivant :

```
Initialisation
#Invariant
while condition :
  Itération
  #Invariant
#Objectif : Invariant et non condition
```

Rappel

Etats, variant et terminaison

- Etats
- Variant

Méthologie complète : exercices

Notion d'état

Pour observer l'exécution d'un programme nous introduisons la notion d'état. Toute action a (instruction) est observée à travers son état avant et son état après.

```
x = 0
# Etat avant : x vaut 0
x = x + 1
# Etat après : x vaut 1
```

En généralisant :

```
# Etat avant : x vaut n
x = x + 1
# Etat après : x vaut n+1
```

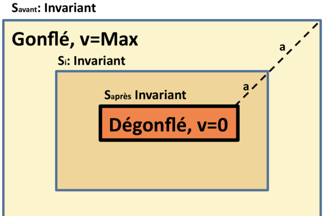
Séquence d'états dans une boucle

```
initialisation
# S_0 avant : la propriété P est vraie
while condition :
  action/instruction
  #S_i : P est vraie dans la séquences S_0...S_i
# S_n : la propriété P est vraie
```

Vérifier la terminaison revient à démontrer que la séquence d'états n'est qu'une suite finie convergente.

- Une boucle qui a pour objectif de dégonfler complètement un ballon (complètement gonflé initialement)
- A chaque itération (action), on laisse sortir une ou plusieurs unités de volumes d'air du ballon
- L'invariant couvre tous les états y compris les états S_{avant} à S_{apres} : `gonfle(volume) and 0<=volume<=maximum`
- Condition d'arrêt : `volume==0`

Le volume ($\in [0..maximum]$) se réduit (d'au moins une unité) à chaque itération, on peut donc garantir la terminaison à l'aide d'une suite finie convergente (la suite des volumes successifs).



Identifier un variant

Une suite finie convergente est souvent exprimée par une valeur entière, qu'on appelle variant, qui doit satisfaire :

- variant $>$ borne inférieure
- variant décroit à chaque itération

Instrumentation pour terminaison en introduisant `t` :

```
Initialisation
#Invariant
while condition:
      # variant borné
      t = variant
      Itération
      #Invariant
      # t > variant
#Objectif : Invariant et non condition
```

Retour à la division euclidienne pour la terminaison

```
X = int(input())
Y = int(input())
q = 0
r = X
assert X==Y*q +r and r>=0, "erreur initialisation"
while (r>=Y):
    assert r>0, "erreur variant"
    t = r
    q = q + 1
    r = r - Y
    assert X==Y*q +r and r>=0, "erreur iteration"
    assert t>r, "erreur terminaison"
assert X==Y*q +r and r>=0 and r<Y, "erreur resultat"
print("q=",q)
print("r=",r)
```

Entre le début de boucle et la fin du corps de la boucle, r doit avoir changé (diminué).

Exercice : Rédiger un assert

Instrumenter le code pour la terminaison

- Ecrivez les 3 lignes (avec ou sans assert) et leur position qui permettent de tester la terminaison de la boucle.

```python
liste = eval(input())

somme = 0
i=0

while i<len(liste):


    somme = somme + liste[i]
    i = i+1


print(somme)
```

Rappel : ces instrumentations se font théoriquement AVANT l'écriture de la boucle

Correction assert somme

```
liste = eval(input())
somme = 0
i=0
while i<len(liste):
  assert len(liste)-i>0, "erreur variant"
  t = len(liste) - i
  somme = somme + liste[i]
  i = i+1
  assert t> len(liste)-i ,"erreur terminaison"
print(somme)
```

- Trouver le variant et ajouter les assert correspondant pour l'exercice min/max vu précédemment

```
L = eval(input())
if L[0]<L[1] :
    imin , imax=0,1
else:
    imin , imax=1,0
i=2
assert 0 <= imin < i <= len(L) and imin!=imax and 0<= imax < i <= len(L)\
  and L[imin]==min(L[0 :i]) and L[imax]==max(L[0:i]), "erreur init "
while i!= len(L):
    ...
    ...
    if L[ i ]>L[imax]:
        imax=i
    elif L[j] <L[imin]:
        imin=i
    i =i+1
    print(imin,i,len(L))
    assert 0 <= imin < i <= len(L) and imin!=imax and 0<= imax < i <= len(L)\
      and L[imin]==min(L[0 :i]) and L[imax]==max(L[0:i]), "erreur iter "
    ...

assert 0 <= imin < len(L) and imin!=imax and 0<= imax < len(L) and L[imin]==min(L) and
       L[imax]==max(L), "erreur objectif"
print (" indice du min de L=", imin)
print (" indice du Max de L=", imax)
```

Correction min/max

```python
L = eval(input())
if L[0]<L[1] :
    imin , imax=0,1
else:
    imin , imax=1,0
i=2
assert 0 <= imin < i <= len(L) and imin!=imax and 0<= imax < i <= len(L)\
and L[imin]==min(L[0 :i]) and L[imax]==max(L[0:i]), "erreur init "
while i!= len(L):
    assert len(L)-i>0, "erreur variant"
    t = len(L) - i #variant
    if L[ i ]>L[imax]:
        imax=i
    elif L[j] <L[imin]:
        imin=i
    i =i+1
    print(imin,i,len(L))
    assert 0 <= imin < i <= len(L) and imin!=imax and 0<= imax < i <= len(L)\
    and L[imin]==min(L[0 :i]) and L[imax]==max(L[0:i]), "erreur iter "
    assert t>len(L)-i, "erreur terminaison"
assert 0 <= imin < len(L) and imin!=imax and 0<= imax < len(L) and L[imin]==min(L) and
      L[imax]==max(L), "erreur objectif"
print (" indice du min de L=", imin)
print (" indice du Max de L=", imax)
```

Rappel

Etats, variant et terminaison

Méthologie complète : exercices

- Objectif et Terminaison
- Pivot
- Drapeau Hollandais

Pour résoudre les exercices suivants :

- a) avec l'aide de l'enseignant, identifier et préciser l'objectif
- b) selon le modèle de solution imposé, obtenir à partir de l'objectif : l'invariant et la condition d'arrêt
- c) déterminer le variant
- d) écrire le code

```
Initialisation
#Invariant
while condition:
      # variant borné
      t = variant
      Itération
      #Invariant
      # t > variant
#Objectif : Invariant et non condition
```

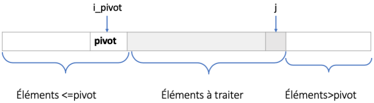
Exercice pivot

Considérons un tableau (représenté par une liste) de n entiers (n>0). Il s'agit de placer un élément du tableau (appelé **pivot**) à une place définitive en permutant tous les éléments de telle sorte que :

- tous les éléments qui sont inférieurs ou égaux au pivot soient à sa gauche,
- tous les éléments qui lui sont supérieurs soient à sa droite.

Pour partitionner le tableau :

- ici, on prend le premier élément comme pivot,
- on doit établir une partition en trois. Ceci nécessite le développement d'une boucle dont l'objectif est de répartir les éléments du tableau dans les deux parties extrêmes
- on ne parcourt le tableau qu'une fois,



Pivot

Pour exprimer les propriétés à vérifier, on utilisera les fonctions `est_sup_eq_max` et `est_inf_min`. Elles renvoient True si x est supérieur (resp. inférieur) au maximum (resp. minimum) de la liste (même si elle est vide).

```python
# opérations utilisées uniquement lors de la mise au point du programme dans des assert

def est_sup_eq_max(x,liste) : #renvoie True si x est supérieur ou égal au maximum de la liste
    if liste==[] :
        return True
    return x>=max(liste)

def est_inf_min(x,liste) :#renvoie True si x est inférieur au minimum de la liste
    if liste==[] :
        return True
    return x < min(liste)
```

Exercice pivot

- Donner l'état final de l'algorithme pour l'exemple
tableau = [1, -9, 7, 4, 2, -1, 6, 3, -6, -4, 3, 2, -1]
Attention à l'ordre des éléments "supérieurs"...
- En vous inspirant du graphique, représentez l'état final avec les variables utilisées.
- Exprimez l'objectif en utilisant les fonctions max et min, décomposez le en un invariant et condition d'arrêt
- Exprimez un variant
- Décrire à chaque étape de l'algorithme ce qu'on doit faire avec la valeur située à l'indice i_pivot+1
- Ecrire le code avec les asserts correspondants

Exercice pivot

- Donner l'état final de l'algorithme pour l'exemple
  tableau = [1, -9, 7, 4, 2, -1, 6, 3, -6, -4, 3, 2, -1]

```
tableau = [-9, -1, -4, -6, -1, 1, 3, 6, 2, 3, 2,
    4, 7]
```

ATTENTION a l'ordre des elements superieurs

Exercice pivot

- Donner l'état final de l'algorithme pour l'exemple
  tableau = [1, -9, 7, 4, 2, -1, 6, 3, -6, -4, 3, 2, -1]

```
tableau = [-9, -1, -4, -6, -1, 1, 3, 6, 2, 3, 2,
    4, 7]
```

  ATTENTION a l'ordre des elements superieurs

- En vous inspirant du graphique, représentez l'état final avec les variables utilisées.

On introduit donc deux variables i_pivot et j avec 0<=i_pivot<=j<len(tableau) qui délimitent les éléments qui restent à traiter. Ces deux variables permettront aussi de parcourir le tableau :

- L'arrêt est immédiat quand la partie des éléments à traiter est vide.
- Nous pouvons représenter les trois tronçons de la manière suivante :
  - éléments plus petits : tableau[0 : i_pivot+1]
  - Inconnus : tableau[i_pivot+1 : j+1]
  - éléments plus grands : tableau[j+1 : len(tableau)]

Exercice pivot

- Exprimez l'objectif en utilisant les fonctions max et min, décomposez le en un invariant et condition d'arrêt
  - Invariant :

```
# i_pivot et j dans la tableau
# l'element tableau[i_pivot] est superieur ou egal aux elements entre 0 et i_pivot
# l'element tableau[i_pivot] est inferieur aux elements entre j+1 et la fin du tableau
0<= i_pivot<= j<len(tableau) \
and \
est_sup_eq_max(tableau[i_pivot],(tableau[0:i_pivot+1])) \
and \
est_inf_min(tableau[i_pivot],tableau[j+1:len(tableau)])
```

  - Arrêt : quand la zone "à traiter" est vide :

```
i_pivot == j
# et donc la condtion c'est :
while j != i_pivot :
```

Exercice pivot

- Exprimez un variant : on déduit de la condition d'arrêt que le variant vaut `j-i_pivot`

```
t=j-i_pivot #en debut de boucle
...
assert t >j-i_pivot, "erreur terminaison" # en fin de boucle
```

Exercice pivot

- Exprimez un variant : on déduit de la condition d'arrêt que le variant vaut `j-i_pivot`

```
t=j-i_pivot #en debut de boucle
...
assert t >j-i_pivot, "erreur terminaison" # en fin de boucle
```

- Décrire chaque étape de l'algorithme

```
#si  pivot >  premier élément qu'il reste àtraiter (i_pivot+1) :
#   on les échange et on avance l'indice du prochain élément à traiter
#sinon :
#   si  pivot == premier élément qu'il reste àtraiter (i_pivot+1) :
#      on avance l'indice du prochain élément àtraiter
#   sinon :
#      on échange l'élément restant àtraiter juste avant la zone des éléments supérieurs (avec l'élément en j)
#      on agrandit la zone des éléments supérieurs (j = j-1)
```

Correction Pivot : instrumenté avec assert

```
tableau=eval(input())
i_pivot = #TODO
j= #TODO
assert 0<=i_pivot and i_pivot<= j and j<len(tableau) and est_sup_eq_max(tableau[i_pivot],(tableau[0:i_pivot+1])) and est_inf_min(tableau[i_pivot],tableau[j+1 :len(tableau)]), "erreur init"
while #TODO :
  assert j-i_pivot>0 , "variant"
  t=j-i_pivot
  #TODO
  assert 0<=i_pivot and i_pivot<= j and j<len(tableau) and est_sup_eq_max(tableau[i_pivot],(tableau[0:i_pivot+1])) and est_inf_min(tableau[i_pivot],tableau[j+1 :len(tableau)]) , "erreur iteration"
  assert t>j-i_pivot, "erreur terminaison"
assert est_sup_eq_max(tableau[i_pivot],(tableau[0:i_pivot+1])) and est_inf_min(tableau[i_pivot],tableau[j+1 :len(tableau)]), "erreur objectif"
print(i_pivot)
print(tableau)
```

Correction complète pivot

```python
tableau=eval(input())
i_pivot = 0
j=len(tableau)-1
assert 0<=i_pivot and i_pivot<= j and j<len(tableau) and
      est_sup_eq_max(tableau[i_pivot],(tableau[0:i_pivot+1])) and
      est_inf_min(tableau[i_pivot],tableau[j+1 :len(tableau)]), "erreur init"
while j!=i_pivot:
  assert j-i_pivot>0 , "variant"
  t=j-i_pivot
  if tableau[i_pivot]>tableau[i_pivot+1]:
    tableau[i_pivot], tableau[i_pivot+1] = tableau[i_pivot+1],tableau[i_pivot]
    i_pivot=i_pivot+1
  else:
    if tableau[i_pivot]==tableau[i_pivot+1]:
      i_pivot=i_pivot+1
    else:
      tableau[i_pivot+1], tableau[j]= tableau[j],tableau[i_pivot+1]
      j=j-1
  assert 0<=i_pivot and i_pivot<= j and j<len(tableau) and
        est_sup_eq_max(tableau[i_pivot],(tableau[0:i_pivot+1])) and
        est_inf_min(tableau[i_pivot],tableau[j+1 :len(tableau)]) , "erreur iteration"
  assert t>j-i_pivot, "erreur terminaison"
assert est_sup_eq_max(tableau[i_pivot],(tableau[0:i_pivot+1])) and
      est_inf_min(tableau[i_pivot],tableau[j+1 :len(tableau)]), "erreur objectif"
print(i_pivot)
print(tableau)
```

Exercice Drapeau Hollandais

Ce problème correspond à celui du drapeau Hollandais proposé par Dijkstra. Il s'agit de **trier**, dans l'ordre **bleu, blanc, rouge** une liste de valeurs, avec la particularité que les valeurs à trier appartiennent à l'ensemble **E = {bleu, blanc, rouge}**.

- la liste ne contient pas forcément des valeurs des trois couleurs
- parcourir la liste au plus un fois,
- ne déterminer/lire la valeur d'une case qu'une seule fois,
- ne pas utiliser de listes supplémentaires,
- le nombre initial des bleus, blancs et rouges, doit être conservé. L'affectation multiple sera utilisée,
- pour exprimer les propriétés à vérifier, on utilisera la fonction `couleur_unique(liste,couleur)`

Drapeau Hollandais

```python
# opérations utilisées uniquement lors de la mise au point du programme dans des assert
def couleur_unique( liste, couleur):
    unique = True
    i=0
    while i<len(liste) and unique:
        if liste[i]!= couleur:
            unique = False
        i=i+1
    return unique
```

Drapeau Hollandais

- En vous inspirant de l'exercice précédent, représentez le problème avec un graphique comportant 4 partitions de liste
- Pour définir l'invariant, nous aurons besoin de définir 4 intervalles : bleus,blancs,inconnus, rouge
- Exprimez l'objectif puis l'invariant et la condition d'arrêt
- Exprimez un variant
- Décrire à chaque étape de l'algorithme ce qu'on doit faire avec la valeur à traiter
- Ecrire le code avec les asserts correspondants

Drapeau Hollandais : correction

- Au départ : tous les éléments de la liste sont inconnus
- A l'arrivée : [0 bleus [i blancs [j rouges [N
- Arrêt : quand la tranche d'inconnus est vide
- Au départ les tranches, bleu, blanc et rouge sont vides et la tranche inconnus constituent toute la liste
- Invariant et Arrêt :
  - 0<=j<=k<=N : [0 bleus [i blancs [j inconnus [k rouges [N
  - Au départ cette propriété est vraie car les inconnus = toute la liste.
  - Nous veillons au développement que cette propriété reste vraie à chaque itération.
  - L'arrêt, quand la tranche inconnus est vide c-à-d j=k, donc la condition : j != k
- Terminaison : Il suffit de considérer le compteur k-j qui représente la longueur de l'intervalle des inconnus, il est > 0 dans la boucle et il décroit à chaque itération (soit j qui avance soit k qui recule). Il finira par être = 0.

Drapeau Hollandais assert

```
tableau = eval(input())
N = len(tableau)
#TODO
i =
j =
k =
assert 0<=j<=k<=N and couleur_unique(tableau[0:i],'bleu') and
      couleur_unique(tableau[i:j],'blanc') and couleur_unique(tableau[k:N],'rouge')
while #TODO :
    assert k-j>0, "erreur variant"
    t = k-j
    #TODO
    assert 0<=j<=k<=N and couleur_unique (tableau[0:i],'bleu') and couleur_unique
          (tableau[i:j],'blanc') and couleur_unique (tableau[k:N],'rouge')
    assert t > k-j, "erreur terminaison"
assert 0<=j<=N and couleur_unique (tableau[0:i],'bleu') and couleur_unique
      (tableau[i:j],'blanc')and couleur_unique (tableau[j:N],'rouge')
print(tableau)
```

Drapeau Hollandais correction

```python
tableau = eval(input())
N = len(tableau)
i = 0
j = 0
k = len(tableau)
assert 0<=j<=k<=N and couleur_unique(tableau[0:i],'bleu') and couleur_unique(tableau[i:j],'blanc') and couleur_unique(tableau[k:N],'rouge')
while j<k:
    assert k-j>0, "erreur variant"
    t = k-j
    if tableau[j] == 'bleu':
        tableau[i],tableau[j] = tableau[j], tableau[i]
        i=i+1
        j=j+1
    elif tableau[j] == 'blanc':
        j=j+1
    else:
        tableau[k-1],tableau[j] = tableau[j], tableau[k-1]
        k=k-1
    assert 0<=j<=k<=N and couleur_unique (tableau[0:i],'bleu') and couleur_unique (tableau[i:j],'blanc') and couleur_unique (tableau[k:N],'rouge')
    assert t > k-j, "erreur terminaison"
assert 0<=j<=N and couleur_unique (tableau[0:i],'bleu') and couleur_unique (tableau[i:j],'blanc')and couleur_unique (tableau[j:N],'rouge')
print(tableau)
```