Cours Info - 7

Introduction à l'algorithmique

D.Malka

MPSI 2017-2018

Sommaire

Sommaire

Un exemple d'algorithme

Un exemple d'algorithme écrit en *langage naturel*.

Algorithme d'Euclide de recherche du PGCD

Etant donné deux entiers, retrancher le plus petit au plus grand et recommencer jusqu'à ce que les deux nombres soient égaux. La valeur obtenue est le plus grand diviseur commun.

Application

Appliquez l'algorithme d'Euclide aux entiers 133 et 49.

- $133 - 49 = 84$
- $84 - 49 = 35$
- $49 - 35 = 14$
- $35 - 14 = 21$
- $21 - 14 = 7$
- $14 - 7 = 7$
- $7 - 7 = 0$

$PGCD(133, 49) = 7$

Qu'est-ce qu'un algorithme ?

Définition d'un algorithme

Un algorithme est un ensemble d'instructions permettant de résoudre de façon systématique une classe de problème.

Qu'est-ce qu'un programme

Définition d'un programme

Un programme est l'expression d'un algorithme dans un langage de programmation.

Un algorithme en pseudo-code

Algorithme 1 : Algorithme d'Euclide

Entrées : int n, int m

Sorties : int pgcd

```
1 tant que n-m ≠ 0 faire
2 |   si n>m alors
3 |   |   n=n-m
4 |   sinon
5 |   |   m=m-n
6 retourner n /*ici pgcd=n=m*/
```

Un algorithme en Python

L'algorithme d'Euclide implémenté en Python :

```
1  def rech_pgcd(n,m):
2      while n!=m:
3          if n>m:
4              n=n-m
5          else:
6              m=m-n
7      return n
```

Un algorithme en C

L'algorithme d'Euclide implémenté en C :

```
1  int rech_pgcd(int n,int m){
2      while(n!=m){
3          if(n>m){
4              n=n-m
5          }
6          else{
7              m=m-n
8          }
9      }
10     return n
11 }
```

Qu'attend-on d'un algorithme ?

D'un algorithme (a fortiori d'un programme), on attend :

- ***Correction*** **: le résultat doit être celui attendu !**
- ***Rapidité*** **: l'exécution doit être rapide. Notion de *complexité algorithmique en temps*.**
- ***Economie*** **: utilisation minimale de la mémoire vive. Notion de *complexité en espace*.**

Spécification minimale(iste) d'un algorithme

Lors de la recherche d'un algorithme, on commence par une phase de *spécification*.

Spécification minimale(iste) d'un algorithme

- **Entrées et leurs types,**
- **Sorties et leurs types,**
- **Ce que fait l'algorithme (*docstring* des programmes).**

Sommaire

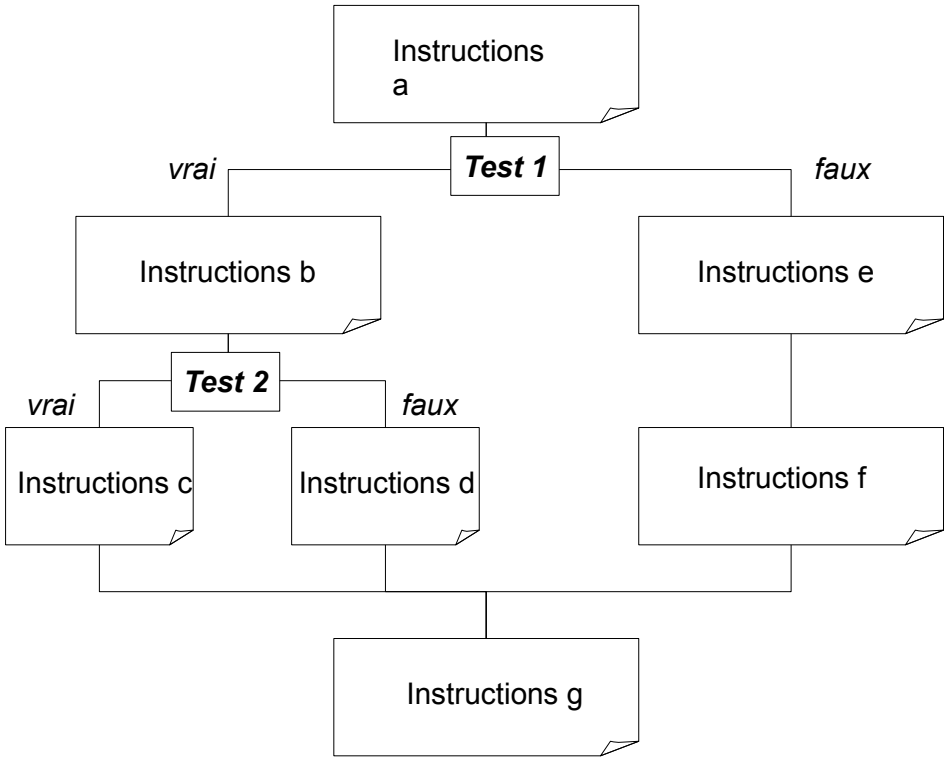
Langage minimal de l'algorithmique

Les 5 instructions suivantes suffisent à exprimer tous les algorithmes imaginables (thèse de Church-Turing) :

- **les instructions simples :**
  - **la déclaration de variable,**
  - **l'affectation de variable,**
- **les instructions composées :**
  - **la séquence (plus généralement le bloc) d'instructions**
  - **le test ou instruction conditionnelle,**
  - **la boucle ou instruction itérative.**

L'embranchement conditionnel

On souhaite contrôler le flot des instructions exécutées suivant la valeur d'une expression (test) : *embranchement conditionnel*.



Structure de l'embranchement conditionnel

L'embranchement conditionnel s'opère grâce à des tests.

Algorithme 2 : Structure de l'embranchement conditionnel

```
1  si test 1 alors
2  |   bloc d'instructions 1
3  sinon si test 2 alors
4  |   bloc d'instructions 2
5  sinon si test 3 alors
6  |   bloc d'instructions 3
7  .
8  .
9  .
0  sinon
1  |   bloc d'instructions final
```

Un exemple simple

Simulation d'un lancer de pièce.

Algorithme 3 : Simulation d'un lancer de Pièce

Entrées : Entier n

Sorties : String res

```
1 p=alea(1,10)
2 si p ≤ 5 alors
3 |   res="Pile"
4 sinon
5 |   res="Face"
6 retourner res
```

Un exemple simple

Implémentation en langage Python : `if... else`.

```
1 def lancer_piece():
2     p=randint(1,10)#renvoie pseudo-aleatoirement un entier entre 1 et
          10.
3     if p<=5:
4         res="Pile"
5     else :
6         res="Face"
7     return res
```

Blocs d'instructions en Python

En langage Python, les blocs d'instructions sont identifiés par l'indentation. Deux instructions de même niveau d'indentation appartiennent au même bloc.

Test avec alternatives indépendantes

Si plus de deux alternatives indépendantes : mot clé `elif`.

```
1 if age<2:
2   print("Bebe!")
3 elif age>2 and age<12:
4   print("Enfant!")
5 elif age>12 and age<25:
6   print("Ado!")
7 else:
8   print("Adulte!")
```

Imbrication de tests

On peut imbriquer les tests :

Algorithme 4 : Structure de l'embranchement conditionnel

```
1  si test 1 alors
2  |   bloc d'instructions 1
3  sinon si test 2 alors
4  |   bloc d'instructions 2
5  sinon
6  |   bloc d'instructions final
7  |   si test a alors
8  |   |   bloc d'instructions a
9  |   sinon
0  |   |   bloc d'instructions b
```

Test avec alternatives

Si plus de deux alternatives non indépendantes : tests imbriqués.

```
1  if voie='agglomeration':
2      if vitesse>50:
3      exces=True
4      else:
5      exces=False
6  elif voie='route':
7      if vitesse>90:
8      exces=True
9      else:
10     exces=False
11 else: #voie=="autoroute"
12     if vitesse>130:
13     exces=True
14     else:
15     exces=False
```

Test avec alternatives

On peut souvent structurer le programme de façon à éviter une imbrication de tests.

```
1 if voie='agglomeration'and vitesse>50:
2     exces=True
3 elif voie='route'and vitesse>90:
4     exces=True
5 elif voie=="autoroute" and vitesse>130:
6     exces=True
7 else:
8     exces=False
```

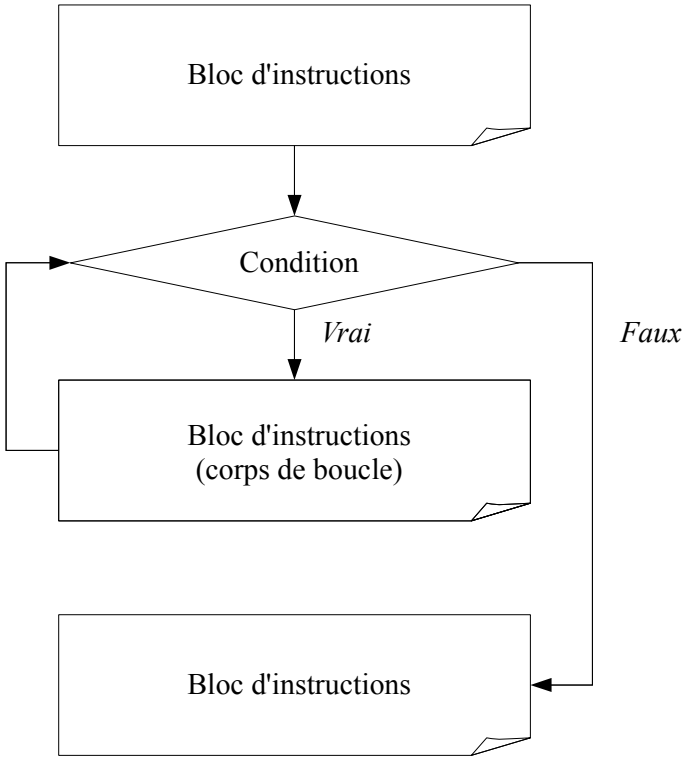
Pourquoi la boucle ?

Pourquoi la boucle ?

La *boucle* ou *instruction itérative* sert à répéter un nombre entier de fois la même instruction.

- **La boucle inconditionnelle : arrêt de la boucle après un nombre fixé d'itérations,**
- **la boucle conditionnelle : arrêt de la boucle suivant le résultat d'un test.**

la boucle


Bloc d'instructions
Condition
Vrai
Faux
Bloc d'instructions
(corps de boucle)
Bloc d'instructions

Structure de la boucle inconditionnelle

Syntaxe :

Algorithme 5 : Structure de la boucle inconditionnelle

```
1 pour i de valeur initiale à valeur finale faire
2 |   bloc d'instructions
  |   /*Corps de boucle*/
```

- **valeur initiale : valeur initiale entière de i**
- **valeur finale : valeur finale entière de i**
- **i est le compteur de boucle**
- **chaque exécution du corps de boucle est appelée *itération***
- **i passe de valeur initiale à valeur finale par incrémentation d'une unité à chaque itération**

Un exemple - Multiples de k

On souhaiter afficher les 10 premiers multiples de k.

Algorithme 6 : Affichage des multiples de k

```
1 pour i de 1 à 10 faire
2 |   afficher k × i
```

Un exemple – Multiples de k

Implémentation en Python :

```
1 for i in range(11):
2     print(k*i)
```

La fonction `range(n,m)` crée une liste qui contient tous les entiers de l'intervalle $[n, m[$.
La fonction `range(n)` crée une liste qui contient tous les entiers de l'intervalle $[0, n[$ soit n valeur.

Exemple – la suite de Fibonacci

Suite de Fibonacci : définition mathématique.

$$\begin{cases} u_0 = 0 \\ u_1 = 1 \\ u_n = u_{n-1} + u_{n-2}, \text{si } n \geq 2 \end{cases}$$

Exemple – la suite de Fibonacci

Suite de Fibonacci :

Spécification

- **ALGORITHME : calcule et renvoie les N+1 premiers termes de la suite de Fibonacci**
- **ENTREE : nombre de termes à calculer N+1**
- **SORTIE : liste des N+1 premiers de termes de la suite**

Algorithme 7 : Suite de Fibonacci

```
1 fibo=[0,1]
2 pour i de 2 à N faire
3 |   u=fibo[i-1]+fibo[i-2]
4 |   ajouter u à fibo
5 retourner fibo
```

Exemple – la suite de Fibonacci

Implémentation en Python :

```
1  def fibonacci(N):
2
3      fibo=[0,1]
4      for i in range(2,N+1):
5          next=fibo[i-1]+fibo[i-2]
6          fibo.append(next)
7      return fibo
```

Ecriture une boucle conditionnelle

Ecrire correctement une boucle inconditionnelle :

1. **Déterminer combien de fois doit s'exécuter la boucle.**
2. **Choisir une variable jouant le rôle de compteur.**
3. **Déterminer si le compteur intervient dans le corps de boucle.**
4. **Ecrire le corps de boucle.**
5. **Prévoir une initialisation des variables en amont.**

Boucle imbriquées

Exemple : afficher les tables de multiplication de 1 à 9

```
1 for i in range(1,10):
2     for j in range(1,10):
3         print(i*j,end=" ")
4     print("\n")
```

Structure

Syntaxe de la boucle conditionnelle :

Algorithme 8 : Boucle conditionnelle

```
1 tant que test faire
2 |   bloc d'instructions
  |   /*Corps de boucle*/
```

Chaque passage dans le corps de boucle s'appelle *itération*.

Syntaxe en langage Python :

```
1 while test:
2     blocs d'instructions
```

Calcul de 2^n

Algorithme qui calcule k^n :

Spécification

- **ALGORITHME : calcule k^n**
- **ENTREES : un entier relatif `k` et un entier naturel `n`**
- **SORTIE : un entier relatif `p`**

Algorithme 9 : Calcul de k^n

```
1 puissance=1 /*valeur si n=0*/
2
3 tant que n>0 faire
4 |   p=p*k
5 |   n=n-1
6 retourner p
```

Boucle conditionnelle

Calcul de k^n

Implémentation en Python :

```
1  def puissance(k,n):
2      p=1 #=k^0
3      while n>0:
4          p=p*k
5          n=n-1
6      return p
```

Tester ce programme sur `http://www.pythontutor.com` et observer comment varie les valeurs des variables.

Boucle conditionnelle

Boucle infinie

Reprenons le calcul de la puissance nième de k :

Algorithme 10 : Calcul de k^n

```
1 puissance=1
  /*valeur si n=0*/
2 tant que n≠0 faire
3 |   p=p*k
4 |   n=n-1
5 retourner p
```

Si n est initialement négatif, la condition d'entrée dans la boucle reste toujours valide : BOUCLE INFINIE !

Boucle conditionnelle

Ecriture correcte

Comment écrire une boucle conditionnelle ?

1. **Identifier la condition de la boucle : souvent la négation de la condition de sortie.**
2. **Une instruction dans le corps de boucle doit modifier la valeur de la condition pour certaines itérations.**
3. **Initialisation : l'initialisation des variables en amont de la boucle est capitale pour la bonne exécution de la boucle.**

Boucle conditionnelle ou inconditionnelle ?

Choix

Si on connaît à l'avance le nombre d'itérations :

$\Rightarrow$ boucle inconditionnelle.

Si on ne connaît pas à l'avance le nombre d'itérations (résultat d'un test) :

$\Rightarrow$ boucle conditionnelle.

Boucle conditionnelle ou inconditionnelle ?

Boucle for avec boucle while

On peut toujours réaliser une boucle `for` avec une boucle `while`.
L'algorithme :

Algorithme 11 : Boucle for

```
1 pour i de valeur initiale à valeur finale faire
2 |   corps de boucle
```

. . . peut toujours s'écrire :

Algorithme 12 : Boucle while

```
1 i=0
2 tant que i<=valeur finale faire
3 |   corps de boucle
4 |   i=i+1
```