Algorithimique et programmation

Timothée Poisot

October 16, 2013

Contact

Bureau B-010
Poste 1968
Email: timothee_poisot@uqar.ca

Outils suggérés

- R
- RStudio, TextMate, gEdit, vim, Notepad++, ...
- **papier et crayon**

Objectifs du cours

1. Développer des bonnes habitudes!
2. Comprendre la structure des programmes
3. Organiser des projets de grande taille

Plan du cours

1. Introduction à l'algorithmique (aujourd'hui)
2. Vecteurs et opérateurs logiques (16 octobre)
3. Stratégies d'analyse de données (30 octobre)
4. Structure du code (13 novembre)

Matériel

```
git clone git@github.com:tpoisot/r_boreas_fall2013.git
```

Ou https://github.com/tpoisot/r_boreas_fall2013

Approche

1. Décomposer le problème
2. Comprendre l'articulation des différents composants
3. Implémenter

Objectifs de la séance

1. Comment penser comme une machine
2. Les étapes de la conception du programme
3. Concevoir et implémenter un algorithme de tri

La *grande* question

1. Qu'est-ce qui rentre?
2. Qu'est-ce qui se passe?
3. Qu'est-ce qui sort?

Avant de commencer

1. Tout va dans des fichiers `.r`
2. Tout va dans des fonctions
3. On commente le code!

Rappel sur les fonctions

```
ma_fonction = function(ce, qui, rentre)
{
   ce qui se passe
   return(ce_qui_sort)
}
```

Exemple

On veut élever un nombre au carré

Entrée: un nombre x
Sortie: x^2
Ce qui se passe: $x \to x^2$

Exemple - en R

On définit la *fonction* `squared`:

```
squared = function(x) {
    return(x^2)
}
```

Puis on teste:

```
print(squared(2))
```

```
## [1] 4
```

```
print(squared(4))
```

```
## [1] 16
```

Notation racourcie

```
squared = function(x) x^2
```

Par défaut, R renvoie *toujours* le résultat de la dernière opération.

Par souci de clarté, on essaie de ne *jamais* utiliser cette propriété

Algorithmes?

Une suite **finie** *et* **non-ambigue** *d'opérations permettant de résoudre un problème*

1. *finitude*: se termine après un nombre fini d'étapes
2. *précision*: étapes définies sans ambigüité
3. *entrées*: on connaît la nature des entrées
4. *sorties*: on connaît leur nature *et* leur relation aux entrées
5. *rendement*: **toute** opération doit être suffisamment simple pour être réalisée à la main en un temps fini

PGCD

L'algorithme d'Euclide: plus grand commun diviseur

Entrée: deux entiers a et b, tels que $a > b > 0$
Sortie: un entier c, tel que $\text{pgcd}(a, b) = c$

PGCD - pseudo-code

```
DÉBUT: r = reste de a // b
si r = 0
   renvoie a
sinon
   a = b
   b = r
   retourne à DÉBUT
```

PGCD - en R

```
PGCD = function(a, b) {
    # Cf. séance 4...
    if (b == 0) {
        return(a)
    } else {
        return(PGCD(b, a%%b))
    }
}
```

PGCD - test

```
print(PGCD(16, 4))
```

```
## [1] 4
```

```
print(PGCD(16, 3))
```

```
## [1] 1
```

```
print(PGCD(450, 150))
```

```
## [1] 150
```

Exercice court

Écrivez un algorithme qui trouve la valeur maximale d'un vecteur $\mathbf{x}$

Pensez: entrées, sorties, étapes

Exercice court - solution

1. $m = \mathbf{x}_0$
2. pour chaque x dans $\mathbf{x}$, $m = x$ si $x > m$
3. à la fin de $\mathbf{x}$, retourner m

Exercice court - en R

```r
getMax = function(x) {
    m = x[1]
    for (i in x) if (i > m)
        m = i
    return(m)
}
getMax(c(1, 2, 3, 4, 5, 4, 3))
```

```
## [1] 5
```

```r
getMax(c(1, 9, 3, 4, 5, 4, 3))
```

```
## [1] 9
```

La boîte à outils

1. Tests logiques
2. Boucles et structures de contrôle
3. Un brin de logique!

Boucles for

```
x = c(1:3)
for (i in x) print(i)
```

```
## [1] 1
## [1] 2
## [1] 3
```

```
for (k in c(1:length(x))) {
    print(x[k])
}
```

```
## [1] 1
## [1] 2
## [1] 3
```

Boucles while

```
Stop = 3
x = c(1:5)
i = 1
while (x[i] <= Stop) {
    print(x[i])
    i = i + 1
}
```

```
## [1] 1
## [1] 2
## [1] 3
```

Boucles `while` - attention

Une boucle `while` a une *condition de sortie* (`x[i] <= Stop` dans le cas précédent)

C'est **votre travail** de vérifier que cette condition de sortie peut être atteinte

Par exemple, le code `while(i < 3) print(i)` tourne indéfiniment

Sortir des boucles - next

```
x = c(1:5)
for (i in x) {
    if (i == 3)
        next
    print(i)
}
```

```
## [1] 1
## [1] 2
## [1] 4
## [1] 5
```

Sortir des boucles - break

```
x = c(1:5)
for (i in x) {
    if (i == 3)
        break
    print(i)
}
```

```
## [1] 1
## [1] 2
```

Résumé

- `for`, `while`, `if/else` (plus la semaine prochaine)
- entrées, sorties
- décomposer en étapes simples

Exercice

Concevoir et **implémenter** un algorithme de tri

Déroulement

1. On discute des entrées / sorties
2. On discute de l'algorithme
3. Vous l'implémentez

Exercice - (une) solution

```
n = 1
tant que n > 0:
   n = 0
   pour i de 2 à longueur(x):
      si x[i] < x[i-1]:
         a = x[i]
         b = x[i-1]
         x[i-1] = a
         x[i] = b
         n = n + 1
renvoie x
```

Exercice - en R

```
bubblesort = function(x) {
    n = 1
    while (n > 0) {
        n = 0
        for (i in c(2:length(x))) {
            if (x[i] < x[i - 1]) {
                a = x[i]
                b = x[i - 1]
                x[i] = b
                x[i - 1] = a
                n = n + 1
            }
        }
    }
    return(x)
}
```

Exercice - test

```
x_1 = c(1, 4, 3, 2, 5, 6)
x_2 = c(6, 5, 4, 3, 2, 1)
bubblesort(x_1)
```

```
## [1] 1 2 3 4 5 6
```

```
bubblesort(x_2)
```

```
## [1] 1 2 3 4 5 6
```

Exercice - remarques

Cet algorithme s'appelle *bubblesort* – c'est une façon *simple* mais *peu efficace* de trier des nombres

Pensez à ce qui arrive si le vecteur `x` est trié de manière décroissante

Exercice - remarques

```
N = 500
v1 = c(1:N)  # already sorted (best case)
v2 = sample(v1)  # randomized (average case)
v3 = rev(v1)  # in inverse order (worse case)
system.time(bubblesort(v1))

##    user  system elapsed
##   0.003   0.000   0.003

system.time(bubblesort(v2))

##    user  system elapsed
##   3.227   0.000   3.223

system.time(bubblesort(v3))

##    user  system elapsed
##   5.030   0.000   5.026
```

Exercice - améliorations?

- il faut du temps pour qu'un grand élément remonte en haut du tableau
- un exemple d'algorithme plus efficace: *quicksort*

Exercice avancé - pour ceux qui le souhaitent

On a deux cercles dont on connaît les centres, et les diamètres

On veut mesurer *approximativement* la surface de l'aire partagée par ces deux cercles – écrivez cet algorithme

Rappel: un point est dans un cercle si la distance entre ce point et le centre du cercle est $\leq$ au diamètre du cercle

Si vous bloquez: "Aiguille de Buffon"