Débogage et profilage de code R

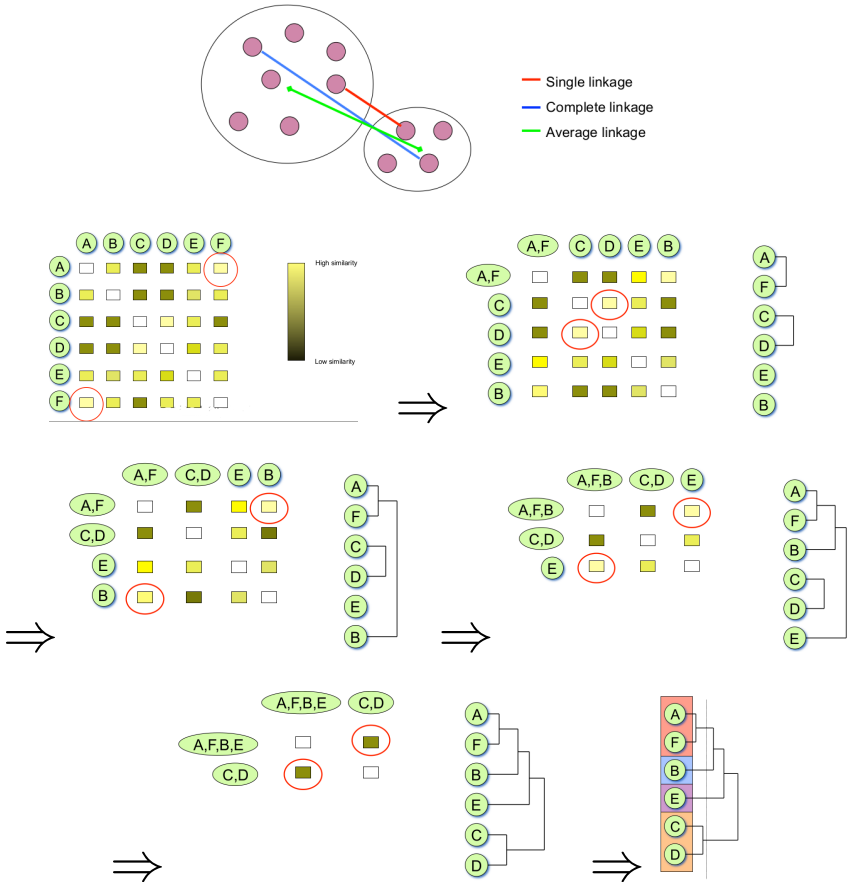
ANF R

Vincent Miele
CNRS

08/10/2015

Ce TP nécessite l'installation préalable des packages `microbenchmark,pryr,igraph`. Avant chaque exercice, il est préférable de relancer `R` (sans sauvegarde de l'environnement).

Exercice 1 (debug avec Rstudio)
(merci de pardonner les anglissimes)
Le package `hclustdemo` propose une implémentation naïve du clustering hiérarchique dont le principe est expliqué dans les figures ci-dessous :



1. Télécharger le package `hclustdemo` sur le site web de l'ANF. Ouvrir un projet Rstudio pour `hclustdemo` et faire un `build`.

2. Exécuter le code de `devtests/testMyHclust.R`

3. Mettre un breakpoint en début de `devtests/testMyHclust.R`, faire un `clean+build` et exécuter ligne à ligne le code avec le menu graphique du debugger. Observer l'évolution des variables dans la fenètre d'environnement.

4. Même chose avec un breakpoint dans `myhclust.R`

5. Il y a un bug dans la fonction `update_clusters`. Avec le debugger, exécuter ligne à ligne le code de cette fonction en observant le contenu de la variable `cluster`

Exercice 2 (sys.time and microbenchmark)

1. Utiliser la fonction `sys.time`.

```
## Book "Advanced R" p.334
x = runif(100)
system.time(
for(i in 1:1e7){sqrt(x)}
)
system.time(
for(i in 1:1e7){x^(1./2.)}
)
```

2. Utiliser le package `microbenchmark` du package et remarquer la précision. Essayer de substituer son utilisation par des appels à `sys.time`.

```
library(microbenchmark)
x=9
microbenchmark(
sqrt(x),
x^0.5,
x^(1./2.),
exp(log(x))/2
)

## Book "Advanced R" p.339
f0 <- function() NULL
f3 <- function(a=1, b=1, c=1) NULL
f6 <- function(a=1, b=1, c=1, d=1, e=1, f=1) NULL
microbenchmark(
f0(),
f3(),
f6()
)

## Book "Advanced R" p.356
mean1 <-  function(x) mean(x)
mean2 <- function(x) sum(x)/length(x)
x <- runif(100)
microbenchmark(
mean1(x),
mean2(x),
mean.default(x)
)
```

Exercice 3 (profiling mémoire)

1. Comprendre le résultat de chaque appel de la fonction `mem_used`.

```
## Book "advanced R" p.379-81
library(pryr)
```

```
mem_used()

v=1:1e8 ## attention !!!
mem_used()
object_size(v)
v[1] = 0
mem_used()

library(pryr)
v=1:1e8 ## attention !!!
mem_used()
w = list(v,v,v)
object_size(w)
mem_used()
v[1] = 0
mem_used()
```

2. Comprendre le résultat de chaque appel de la fonction `tracemem`.

```
library(pryr)
v=1:1e8 ## attention !!!
mem_used()

tracemem(v)
for (i in 1:10){
v[i] = v[i+1]
print(mem_used())
}

tracemem(v)
for (i in 1:10){
v[i] = 63
print(mem_used())
}
```

3. Comprendre le résultat de chaque appel de la fonction `mem_change`.

```
library(pryr)
N=2e7 ## attention !!!
x = runif(N/4)
y = runif(N/4)
mem_used()
mem_change(
points <- cbind(c(x,x+1.1,x,x+1.1),
c(y,y+1.1,y+1.1,y))
)
mem_change(rm(x))
mem_change(rm(y))
```

Exercice 4 (Rprof - découverte 1)
Etudier le résultat de `Rprof` pour profiler le code ci-dessous et découvrir le "hotspot":

```
library(igraph)
stat4graph <- function(g){
```

```
s1 <- independence.number(g)
s2 <- no.clusters(g)
s3 <- mean(betweenness(g))
s4 <- mean(degree(g))
c(s1,s2,s3,s4)
}

Rprof("testRprof.out", interval = 0.005)
graphs <- NULL
for (i in 1:100) graphs[[i]] <- erdos.renyi.game(60, 1/15)
stats <- NULL
for (i in 1:100) stats[[i]] <- stat4graph(graphs[[i]])
mstats <- matrix(unlist(stats), ncol = 4, byrow = TRUE)
apply(mstats, 2, mean)
Rprof()
summaryRprof("testRprof.out")
```

Exercice 5 (Rprof - découverte 2)
Utiliser `Rprof` pour profiler le code ci-dessous et découvrir le "R inferno":

```
grow.method <- function(n){
v <- numeric(0)
for(i in 1:n) v <- c(v,i)
v
}
subscript.method <- function(n){
v <- numeric(n)
for(i in 1:n) v[i] <- i
v
}
operator.method <- function(n){
v <- 1:n
}

n=1e5
system.time({
u <- grow.method(n)
})
system.time({
u <- subscript.method(n)
})
system.time({
u <- operator.method(n)
})

## Learning Rprof
Rprof()
u <- grow.method(n)
u <- subscript.method(n)
u <- operator.method(n)
Rprof(NULL)
summaryRprof()
```

Exercice 6 (Rprof - analyse complète)
Utiliser `Rprof` pour profiler étudier les performances du package `hclustdemo` :

1. A partir de `devtests/testMyHclust.R`, réaliser le profiling de la fonction `my.hclust` avec Rprof. Trouver le "hotspot".

2. Remplacer la fonction ad-hoc par son équivalent en C++ disponible dans `src`. Faire un `clean+build` et refaire le profiling.