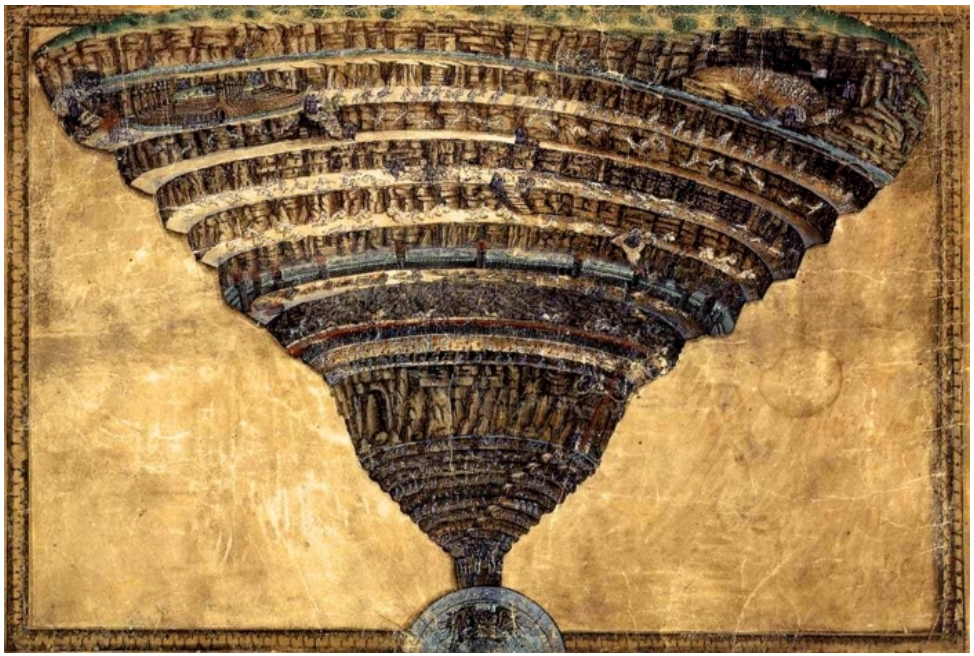
Ад R.

Алексей Осипов, к.ф.-м.н.
osipovav28@googlemail.com

Сиденис

19 октября 2018

Структура ада.



Гигантская воронка. 9 кругов.

Первый круг.

Язычники. Люди, не догадывающиеся о проблемах, связанных с округлением чисел.

Первый круг, детали.

Ввод	На экране
`print(0.4-0.1)`	0.3
`print((0.4-0.1)==0.3)`	FALSE
`print((0.4-0.1)-0.3)`	5.551115e-17

Решение:

Ввод	На экране
`print(all.equal(0.4-0.1, 0.3))`	TRUE
`print(abs((0.4-0.1)-0.3)<0.00001)`	TRUE

Схожие проблемы есть и в других языках программирования.

Второй круг.

Lust/*жадность*. Заполнение вектора вместе с увеличением его размера в цикле в лоб, что в итоге **очень** медленно.

Второй круг, детали.

1. Метод 1.

```
v <- numeric(0)
for (i in 1:n) {v <- c(v, i^2)}
```

Время работы: 5422.21 секунды, на n = 1000000.

2. Метод 2.

```
v <- numeric(n)
for (i in 1:n) {v[i] <- i^2}
```

Время работы: 0.08 секунды, на n = 1000000.

3. Метод 3.

```
v <- (1:n)^2
```

Время работы: меньше 0.01 секунды, на n = 1000000.

Третий круг.

Обжорство/*сребролюбие*. Недостаточная векторизация кода.

Третий круг, детали.

1. Метод 1.

```
v <- 1:n
s <- 0
for (i in 1:n) {
  if (v[i] %% 2 == 0) {
            s <- s + log(v[i])}}
```

Время работы: 0.42 секунды, на n = 1000000.

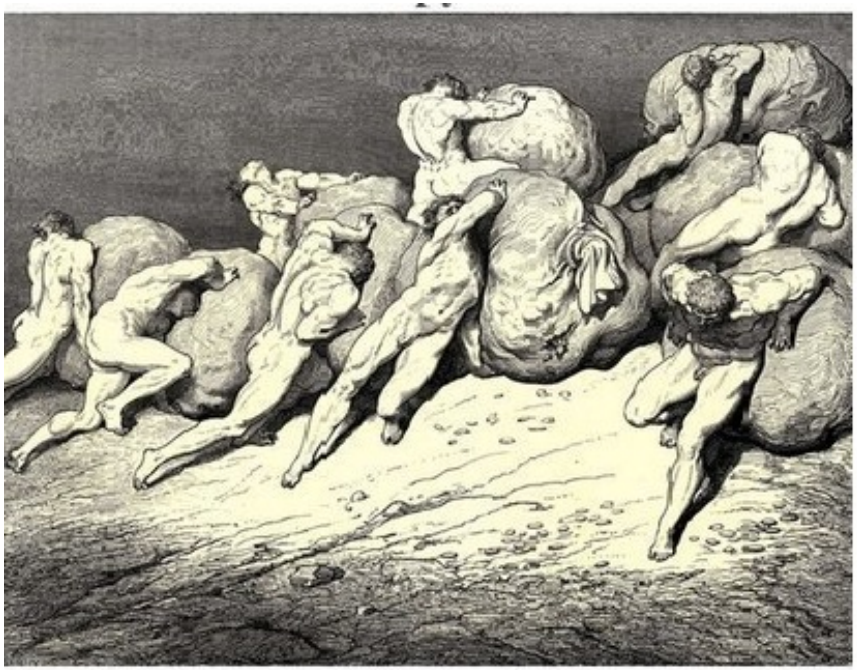
2. Метод 2.

```
v <- 1:n
s <- sum(ifelse(v %% 2 == 0, log(v), 0))
```

Время работы: 0.17 секунды, на n = 1000000.

Дело не в скорости, а в качестве кода.

Четвертый круг.



Избыточная привязанность к деньгам. Излишняя векторизация кода.

Четвертый круг, детали.

- В R есть функции семейства **apply**, фактически применяющие функцию к какой-то структуре данных (вектор, матрица, ...). По скорости они быстрее цикла, но не слишком. Насколько интенсивно использовать их комбинации зависит от того, какой код легче читать и поддерживать.
- Бывает, что излишняя векторизация забирает много памяти. Пусть x — дата-фрейм (таблица) с **большим** количеством столбцов.

Метод 1.

```
x[is.na(x)] <- 0
```

Метод 2.

```
for (i in 1:ncol(x)) {x[is.na(x[, i]), i] <- 0}
```

Пятый круг.

Гнев/*измена*. Отказ от написания функций.

Пятый круг, детали.

- Функции нужны для абстракции, воспроизводимости, есть безымянные функции.
- Ручная обработка особых случаев (NA, вектора длины 0, ...) — плохая идея.
- Избегать побочных эффектов функций.
- Входными и выходными параметрами функции могут быть функции.

```
mycumfun <- ecdf(rnorm(10))
```

- Правильно планировать типы аргументов (самые простые возможные).
- Помнить, что аргументы могут быть (стать) векторами.
- Warnings гораздо опасней чем errors.

Шестой круг.

Ересь. Использование глобальных переменных, побочные эффекты функций в другом environment.

Шестой круг, плохой пример.

```
function(x) {
        y <<- 239
        return(x^2)}
```

Совершенно неожиданно при вызове функции возникает глобальная переменная y (или меняет значение уже существующая).

Шестой круг, хороший пример.

```
fibonacci <- local({
        N <- 100; memo <- c(1, 1, rep(NA, N));
        f <- function(x) {
          if (x==0) { return(0)
        } else if (x>N+2) { stop("Case not
          ↪ supported.")
        } else if (!is.na(memo[x])) {return(
          ↪ memo[x])
        } else { memo[x] <<- f(x-2) + f(x-1)
        return(memo[x])}}})
fibonacci(7)
```

Можно посмотреть сохраненные значения:

```
head(get('memo', envir=environment(fibonacci)),
  ↪  10)
```

Седьмой круг.

Насилие. Злоупотребление объектами и классами.

Седьмой круг, обзор.

Объектная система R:

- S3-классы (передаются по значению)
- S4-классы (передаются по значению)
- R5-классы (передаются по ссылке)
- R6-классы (передаются по ссылке)

S3-классы определяются исключительно аттрибутом class. Функции generic на них смотрят и вызывают соответствующий метод: print, print.default, print.lm. Корректность проверяется в процессе выполнения.
Объекты S4-классов состоят из **слотов**, обращение к слоту осуществляется так: x@data, для проведения конвертации нужно соответсвие структур объектов.
Для S3 метод выбирается в процессе выполнения кода, для S4 — при загрузке кода в сессию, но до выполнения.

Восьмой круг.

Мошенники. Самообман.

Призраки. Химеры. Дьяволы.

Восьмой круг, призраки.

- Приоритет операций.

```
1:n-1 == (1:n)-1
```

- x==NA дает NA, поэтому есть **is.na**(x) и **is.null**(x).
- **round**(2.5) == 2, но **round**(3.5) == 4, чтобы устранить statistical bias при округлении.
- Автоматическое приведение типов:

```
50 < '7'
```

- Error: unexpected 'else' in "else"

```
if (a<1) print("Case␣1")
else print("Case␣2")
```

- Пусть mylist = **list** (1,2,3,4,5) . Есть разница между mylist [[1]] (1-й элемент) и mylist [1] (список, содержащий 1-й элемент)

Восьмой круг, химеры.

- Факторы.

```
fac <- factor(c(1, "A", "B", 0, -1.7, "A",
    ↪ "B", 1))
str(fac)
Factor w/ 5 levels "-1.7","0","1",..: 3 4 5
    ↪ 2 1 4 5 3
```

- Логические операции **&&** и || работают со скалярными аргументами (и могут смотреть только на первый аргумент), операции **&** и | работают с векторными аргументами.
- Разница между f(x=5) и f(x<-5) в создании глобальной переменной x.
- Разница между **max**() и **pmax**() (p for parallel) в том, что первая функция возвращает число, а вторая — вектор.

Восьмой круг, дьяволы.

- **read.table** по умолчанию конвертирует строки в факторы (stringsAsFactors = TRUE). Фактор — скорее число, а не строка. Эта конвертация может быть как очень удобна, так и очень неудобна в зависимости от целей.
- Сохранение в файл с помощью **write.table**, а потом чтение с помощью **read.table** может сильно поменять объект. Лучше использовать **save**, **attach**, **load**.
- Случайное использование глобальной переменной:

```
fun <- function(x) x + y
```

- Дефолтные аргументы:

```
fun <- function(x, y=x) x+y
x <- 100
```

Тогда fun(2) дает 4, а fun(2, x) дает 102.

Девятый круг.

Измена/*воровство*. Неуважение других при обращении за помощью.

Типичный пример:

```
rm( list=ls() )
```

Что хорошего в R?

- Структура данных дата-фрейм.
- Методы обработки данных на любой вкус (Rccp как в C++, sqldf как SQL, dplyr, напоминающий java, data.table как R).
- Много статистических и вероятностных методов.
- Много библиотек для визуализации данных на любой вкус (кроме стандартных графиков, ggplot2 с возможностью складывать графики, интерактивные rbokeh, plotly).
- Много некоммерческих пакетов на любой вкус, со всеми плюсами и минусами.
- Код по анализу и обработке данных занимает мало места и пишется быстро.
- Возможность писать узкие места на C.