Victor Alberto Romero

Engenheiro eletrônico e de computação. Mestrando em Engenharia Mecânica na USP. Atualmente vinculado ao laboratório de Acústica e Meio Ambiente (LACMAM). Desenvolve pesquisa em acústica submarinha, especificamente no levantamento e análise de paisagens acústicas.

Exercícios

exec

Proposta trabalho final

Plano A: compareSort

Muitos dos que têm chegado no R, têm sido motivados pela necessidade de processar seus dados e automatizar as tarefas que já vinham desenvolvendo com outras ferramentas. Porém, a grande maioria não tem experiência prévia programando. Depois de ter um domínio razoável do R, para muitas pessoas pode ser interessante aprofundar mais os conceitos formais da programação, especificamente na criação de algoritmos. Existem múltiplas técnicas para ensinar a fazer isto, mas uma das mas divertidas e didáticas é através dos algoritmos de ordenação (por exemplo, é o caminho usado por Cormen em seu célebre "Introduction to Algorithms"). Para isto estou propondo uma função que permita comparar algoritmos de ordenação graficamente para estudá-los. Adicionalmente vão ser criadas algumas funções auxiliares de ordenação (inicialmente estou pensando em 3: quicksort, mergesort e insertionsort) como exemplo.

Função principal:

```
  compareSort(fun, vals, step=0.5, ...)
```

```
  fun -> é o nome da função de ordenação a comparar. Pode ser uma lista de
funções.
  vals -> é um vetor com valores numéricos a serem ordenados.
  step -> é o intervalo entre uma imagem e outra.
  ... -> parâmetros a serem passados às funções de ordenação e a função par
```

A saída esperada da função é um gráfico com uma ou mais animações comparando os algoritmos. Algo parecido com isto:

Funções auxiliares (aplicáveis para todas):

```
quicksort(vals)
retorna um vetor com os valores ordenados pelo método quicksort.
```

Plano B

Fazer uma função que pinte o fractal de koch!!

A função proposta é:

```
kochPlot(n, step, ...)
```

```
n -> é um número como o nível de profundidade desejado do fractal
step -> é o intervalo entre uma imagem e outra.
... -> os parâmetros que vão ser repassados para a função segment
```

> Olá Victor,
>
> Vejo que você escolheu tarefas clássicas de cursos de computação! A animação da comparação das funções de ordenação é muito interessante. Um exercício desses não é típico para um estudo de R, em parte porque a perda de eficiência ao se implementar suas próprias funções de baixo nível em R é enorme em comparação com as funções base, implementadas em C. Sugiro como curiosidade que você faça a comparação de eficiência entre a sua implementação de qualquer sort com o sort do r-base, é impressionante. Outra peculiaridade do R, como costuma acontecer em linguagens de alto-nível, é que a escolha da estrutura de dados determina a performance de operações em cima desses dados. Isso é só um comentário de alguém que se interessa pelo assunto! Agora preciso comentar diretamente as suas escolhas de proposta.
>
> Uma das exigências para esse trabalho é que as funções sejam *genéricas*. Sua proposta B é muito pouco genérica, dado que ela só desenha esse fractal específico. Dessa forma, recomendo que você faça a proposta A, ou alguma variação dela.
>
> A proposta A é bem ambiciosa, dado que pra fazer uma animação dessas você vai precisar guardar a ordenação do vetor a cada operação. Como isso vai ter que ser implementado por você, imagino que o parâmetro “fun” vai ser uma opção dentro de um número bem limitado de opções, não? Sugiro não se preocupar a princípio em plotar muitas funções ao mesmo tempo - na verdade fazer a animação de uma ordenação só de cada vez já seria bom pra começar. Por outro lado, se você só implementar um método de ordenação, a função fica muito pouco genérica. Você poderia incluir algum output em texto também por exemplo, o número total de operações que foi usado para ordenar.
>
> —-*Mali*

Página de Ajuda

```
sortComparing                package:unknown                R Documentation

Compara dois algoritmos de ordenação.

Description:

     A função recebe dois funções funA e funB e um conjunto de números vals
e
     retorna um video em mp4 com a comparação da execução dessas funções
sobre
     o conjunto de números. A função pode ser chamada com qualquer algoritmo
     enquanto ele retorne todos os estados intermediários em forma de
matriz.

Usage:

     compareSort(funA, funB, vals, interval=0.05, width=800, height=600,
...)

Arguments:

    funA: Uma função de ordenação que retorna uma matriz com todos os
estados.

    funB: Uma função de ordenação que retorna uma matriz com todos os
estados.

    vals: Um vetor de valores a serem ordenados

    interval: O intervalo entre cada plot da animação

    width: A largura da animação

    height: A altura da animação

    ...: Parâmetros a serem passados para barplot

Details:

    As funções funA e funB podem ser qualquer função de ordenação da forma:

        funA(vect, stepLog=FALSE)

    Na qual vect e um vetor numérico e stepLog define se a função retorna só
o
    valor final ou os passos intermediários.

Value:
```

```
     A função vai criar um vídeo animation.mp4 com a comparação de funA e
funB

Note:

     A função requer o pacote animagion.

Author(s):

     Victor Alberto Romero
     vialrogo@gmail.com

References:

    https://pt.wikipedia.org/wiki/Insertion_sort
    https://pt.wikipedia.org/wiki/Selection_sort

See Also:

     animation, barplot

Examples:

    compareSort(selectionSort, insertionSort, vals = sample(seq(1,30)),
interval=0.01)

    compareSort(insertionSort, selectionSort, vals = sample(seq(1,100),30),
interval=0.01)
```

Código da Função

```
# insertionSort: Função para ordernar numero ascendentemente mediante o
algoritmo
# insertion sort: https://pt.wikipedia.org/wiki/Insertion_sort
insertionSort <- function(vect, stepLog=FALSE) {
    acc <- c() # matriz que vai contener todos os estados gerados
    for( i in seq(1,length(vect))) { # Para cada posição do vetor
        for(j in seq(1,(length(vect)-1))) { # Para cada uma das outras
posições
            if(vect[j] > vect[i]) { # Comparo se a posição atual é menor que
as outras
                aux <- vect[i] # Guardo o valor numa variavel auxiliar
                vect[i] <- vect[j] # Troco os dos números
                vect[j] <- aux # Recupero o número guardado na variavel
auxiliar
            }
            acc <- c(acc,vect) # Guardo o estado intermediario na matriz
        }
```

```
    }
    acc <- matrix(acc, nrow= length(vect)) # formato a matriz em forma
matriz
    if (stepLog) # Pergunto se quer a saida completa ou só a final
        return(acc) # retorno toda a matriz de saida
    else # se quer só o vetor final
        return(vect) # retorno só o vetor final
}

# selectionSort: Função para ordernar numero ascendentemente mediante o
algoritmo
# selection sort: https://pt.wikipedia.org/wiki/Selection_sort
selectionSort <- function(vect, stepLog=FALSE) {
    acc <- c() # matriz que vai contener todos os estados gerados
    for( i in seq(1,length(vect))) { # Para cada posição do vetor
        minor <- Inf # No começo de cada ciclo, o mínimo é infinito
        minj <- i # a menor posição é a mesma no começo de cada ciclo
        for(j in seq(i,length(vect))) { # Para cada posição MAIS NA FRENTE
            if(vect[j] < minor) { # Se encontro um valor menor
                minor <- vect[j] # Guardo o valor menor encontrado
                minj <- j # Guardo o indice do menor valor encontrado
            }
            acc <- c(acc,vect) # Guardo o estado intermediario na matriz
        }
        aux <- vect[i] # Guardo o valor numa variavel auxiliar
        vect[i] <- vect[minj] # Troco os dos números
        vect[minj] <- aux # Recupero o número guardado na variavel auxiliar
    }
    acc <- matrix(acc, nrow= length(vect)) # formato a matriz em forma
matriz
    if (stepLog) # Pergunto se quer a saida completa ou só a final
        return(acc) # retorno toda a matriz de saida
    else # se quer só o vetor final
        return(vect) # retorno só o vetor final
}

library(animation) # Biblioteca para gerar as animações

# compareSort: Função que recebe duas outras funções de ordenação e um vetor
de
# valores e gera uma animação para comparar elas.
compareSort <- function(funA, funB, vals, interval=0.05, width=800,
height=600,...){
    outA <- funA(vals, TRUE) # Evalua a função A
    outB <- funB(vals, TRUE) # Evalua a função B
    # Define as propriedades da animação
    ani.options(interval=interval, ani.width=width, ani.height=height)
    saveVideo({ # Começa a salvar as imagens para a animação
        nStepsMax <- max(ncol(outA), ncol(outB)) # Calcula qual é mais longa
        par(mfrow=c(1,2)) # define o layout do plot
        for (i in 1:nStepsMax) { # para cada imagem que vai ser gerada
```

```
            ia <- i # que indice vai ser plotado para a primeira imagem
            ib <- i # que indice vai ser plotado para a segunda imagem
            if (i > ncol(outA)) ia <- ncol(outA) # Se terminou, plota o
último
            if (i > ncol(outB)) ib <- ncol(outB) # Se terminou, plota o
último
            barplot((outA)[,ia],...) # plota o estado atual
            barplot((outB)[,ib],...) # plota o estado atual
        }
    })
}
```

Arquivos da Função

help.txt

sortscomparing.r