**PCS3556 - Lógica Computacional**

# 1º Exercício-Programa
## *Introdução à Programação Funcional*

**Professor Doutor Ricardo Rocha**
*Luís Henrique Barroso Oliveira - 9783640*
*Rodrigo Vali Cebrian - 9835623*
*Vanderson da Silva dos Santos - 11259715*

**São Paulo, 19 de fevereiro de 2023**

# Introdução

## Enunciado

O objetivo didático desta atividade é experimentar concretamente os conceitos desenvolvidos em sala de aula a respeito de gramáticas, cadeias, etc. Além disso, deve ser usada uma linguagem funcional como paradigma de linguagem de programação a linguagem Elixir ou Clojure.

O objetivo do exercício é implementar o algoritmo de fecho reflexivo e transitivo de uma relação binária $\boldsymbol{R} \subseteq \boldsymbol{A} \times \boldsymbol{A}$ sobre um conjunto finito A que é descrita por meio de um grafo direcionado. Conforme definido em sala de aula, a solução deverá construída por meio de recursão.

## Entendimento do problema

O enunciado descreve o problema de se obter um fecho reflexivo e transitivo, dados: (1) um conjunto ***A***, e (2) ***R*** consiste de um grafo direcional, cujos nós são elementos de ***A***, ou seja, representando uma relação binária sobre o conjunto ***A***. Por exemplo, havendo em ***R*** a relação ***(a,c)***, significa que existe uma aresta ligando o nó ***a*** até o nó ***c***, com direção definida.

Posto que o algoritmo requisitado é de fecho transitivo e reflexivo, é definem-se os conceitos a seguir, chamando de ***R'*** a relação resultante:

- Fecho ***transitivo***: dados ***b, c, d*** pertencentes a ***A,*** com ***(c,d)*** e ***(d,b)*** pertencentes a ***R***, então ***(c,d)***, ***(d,b)*** e ***(c,b)*** pertencem a ***R'*** .
- Fecho ***reflexivo***: um nó sempre estará direcionado a ele mesmo, ou seja, dado ***a*** pertencente a ***A,*** então ***(a,a)*** pertence a ***R'*** .

O problema apresentado deve ser solucionado utilizando programação funcional. As duas linguagens que podem ser empregadas são ***Clojure*** e ***Elixir***. No escopo da presente solução, optou-se ***Clojure.***

# Solução

O código completo desenvolvido está presente no **Anexo A**.

A lógica principal do código está descrita na ***main*** do código. Para fins de simplificação, em vez de representar os nós do grafo como de **a**, **b**, **c**, etc, optou-se por representá-los como números: **1**, **2**, **3**, etc.

```
(defn -main []
 (def A [1 2 3])
 (println "A:" A)
 (def n (count A))

 (def R
   (CreateR [1, 2, 3]
            [2, 3, 3])
 )
 (println "R:" R)

 (def PossibleReflectiveValues
   (CreateR A A)
 )

 (def ReflectiveR
   ( CreateReflectiveR R PossibleReflectiveValues )
 )
 (println "Fecho Reflexivo  de R | Reflective(R):" ReflectiveR)

 (def TransitiveR
   ( Transitive n R)
 )
 (println "Fecho Transitivo de R | Transitive(R):" TransitiveR)
)
```

A lógica principal consiste em:

1. Carregar o valor dos nós A do grafo
2. Obter o valor quantidade n de nós do grafo
3. Carregar a matriz de transição entre os nós de R
4. Gerar todos os valores possíveis de reflexão

5. A partir desse valores possíveis de reflexão, gerar o fecho reflexivo de R
6. A partir do valor n de nós e da matriz R, gerar o fecho transitivo de R

Como se calcula o fecho reflexivo e transitivo será demonstrado nos tópicos seguintes.

## Fecho Reflexivo

O fecho reflexivo foi mais simples de encontrar. De maneira geral, criou-se uma função que recebe ***R*** e ***todos os valores possíveis de reflexão*** (ou seja ou valores da diagonal principal da matriz). Essa função é chamada na *main* do código.

Em relação ao seu funcionamento, primeiramente gera-se uma função *lambda* chamada ***DataRemoveCondition***, que verifica que o valor de entrada **x** está presente em **x**. Depois disso, emprega-se a função let para criar a variável ***return***, que recebe todos os valores possíveis de reflexão, exceto o que já existe em ***R***. Isso se faz tomando a variável ***PossibleReflectiveValues*** e retirando os valores que já existem em ***R***.

Em seguida, a função retorna o valor de ***return***, além de todos os valores presentes em R.

A função pode ser observada abaixo:

```
(defn CreateReflectiveR[R PossibleReflectiveValues]
 (def DataRemoveCondition (fn [x] (contains? (set R) x)) )
 (let [return (remove DataRemoveCondition PossibleReflectiveValues)]
 (clojure.set/union R return)
 )
)
```

## Fecho Transitivo

O fecho transitivo é bem complicado de implementar que o fecho reflexivo. De maneira geral, o algoritmo consiste em:

1. Tomam-se como valores de entrada n e R
2. A partir dos valores de R, gera-se uma matriz binária com os valores
3. Realizam-se as multiplicações R, $R^2$, $R^3$ ... até Rn ser igual a Rn+1.
4. Depois de realizar essa multiplicação, operam-se as uniões dos valores:

   Fecho transitivo = $RUR^2UR^3U...URn$

$$R^+ = \bigcup_{i \in \{1,2,3,\ldots\}} R^i.$$

Caso algum dos valores dos R seja maior que 1, é considerado 1

5. Depois de calcular a matriz binária do Fecho transitivo e fazer as uniões, transformam-se os valores das posições da matriz final que são 1 em uma lista de coordenadas, e estes valores são retornados.

   A função principal do fecho transitivo está representada abaixo:

```
(defn Transitive [n R]
 (def R_binary (CreateBinaryMatrix n R))
 (def R_final_binary (GetTransitiveR n R_binary R_binary) )
 (def R_final (GetBinatyMatrixIndex n R_final_binary) )
 R_final
)
```

## Recursividade

Para calcular a seguinte equação principal abaixo fecho transitivo, foi utilizada recursividade

$$R^+ = \bigcup_{i \in \{1,2,3,\ldots\}} R^i.$$

De maneira geral, a função ***GetTransitiveR*** tem ***n*** (tamanho da matriz quadrada), ***R*** (matriz de transição R) e ***Rn*** (R multiplicada, toda recursão esse valor aumenta). O algoritmo funciona seguindo os seguintes passos:

1. Recebe n, R e Rn
2. Multiplica R x Rn para gerar Rn+1 (***Rn_plus1***)
3. Se a matriz Rn+1 possuir valores acima de 1, estes são convertidos em 1, impedindo, assim, que os produtos das multiplicações cresçam indefinidamente.
4. Compara-se Rn com Rn+1.
5. Se os valores comparados forem iguais:
   a. Retorna do valor de Rn
6. Se os valores comparados forem diferentes:
   a. Realiza a união de Rn com o retorno da função ***GetTransitiveR*** (n, R, Rn+1)
   b.

Dessa forma, a função é chamada novamente até o momento que a Rn e Rn+1 sejam similares, e, assim, Rn seja retornada.

A função recursiva pode ser vista abaixo:

```
(defn GetTransitiveR [n R Rn]
 (def Rn_plus1 (TranformMatrixInBinaryMatrix n (MatrixMult R Rn)) )
 (if (= Rn_plus1 Rn)
   Rn
```

```
    (MatrixOr n Rn (GetTransitiveR n R Rn_plus1) )
  )
)
```

Para realizar essa função recursiva, outras funções também foram produzidas, como a multiplicação de matriz, a de transformar os valores acima de 1 em 1, etc.A implementação completa de cada uma dessas funções está presente no *Anexo A*, que apresenta o código completo.

# Testes

**Teste 1:** $R$ em uma situação normal

```
vander@vander-Vostro-14-5480:~/Desktop/codes/poli/PCS3556_logica_computacional/ep1$ lein run
A: [1 2 3]
R: [[1 2] [2 3] [3 3]]
Fecho Reflexivo  de R | Reflective(R): [[1 2] [2 3] [3 3] [1 1] [2 2]]
Fecho Transitivo de R | Transitive(R): [[1 2] [1 3] [2 3] [3 3]]
```

**Teste 2:** $R$ vazio, sem relações transitivas a serem cumpridas, somente as reflexivas.

```
vander@vander-Vostro-14-5480:~/Desktop/codes/poli/PCS3556_logica_computacional/ep1$ lein run
A: [1 2 3]
R: []
Fecho Reflexivo  de R | Reflective(R): ([1 1] [2 2] [3 3])
Fecho Transitivo de R | Transitive(R): []
```

**Teste 3:** Grafo com relações transitivas e reflexivas a serem adicionadas.

```
vander@vander-Vostro-14-5480:~/Desktop/codes/poli/PCS3556_logica_computacional/ep1$ lein run
A: [1 2 3]
R: [[1 1] [1 2] [2 3] [3 1]]
Fecho Reflexivo  de R | Reflective(R): [[1 1] [1 2] [2 3] [3 1] [2 2] [3 3]]
Fecho Transitivo de R | Transitive(R): [[1 1] [1 2] [1 3] [2 1] [2 2] [2 3] [3 1] [3 2] [3 3]]
```

**Teste 4:** Grafo mais complexo, com relações transitivas e reflexivas a serem adicionadas.

```
vander@vander-Vostro-14-5480:~/Desktop/codes/poli/PCS3556_logica_computacional/ep1$ lein run
A: [1 2 3 4]
R: [[1 1] [1 2] [2 2] [2 3] [2 4] [3 1] [4 4]]
Fecho Reflexivo  de R | Reflective(R): [[1 1] [1 2] [2 2] [2 3] [2 4] [3 1] [4 4] [3 3]]
Fecho Transitivo de R | Transitive(R): [[1 1] [1 2] [1 3] [1 4] [2 1] [2 2] [2 3] [2 4] [3 1] [3 2] [3 3] [3 4] [4 4]]
```

# Conclusão

Com esta atividade, pode ser observada a diferença na implementação da linguagem funcional e seus benefícios ao lidar com problemas matemáticos. Houve muita dificuldade ao se adaptar à lógica da linguagem funcional Clojure.

A adaptação com linguagem clojure não muito trivial, principalmente pela linguagem não ser muito popular e a documentação não ser muito completa e com exemplos de implementações. Além do material oficial, ainda não existem muitas bibliotecas por fora feitas para serem usadas, assim, muitas funções comuns tiveram que ser feitas do zero.

# Bibliografia

https://pt.wikipedia.org/wiki/Fecho_reflexivo#:~:text=Em%20matem%C3%A1tica%2C%20o%20fecho%20reflexivo,necess%C3%A1rios%20para%20a%20tornar%20reflexiva.

http://amatematicadiscreta.blogspot.com/2015/05/propriedade-de-fecho.html

https://www.cin.ufpe.br/~gdcc/matdis/aulas/relacoes2.pdf

https://github.com/VanderSant/PCS3556_computational_logic

# Anexo A - Código Completo

```
;; Defintion of namespace
-----------------------------------------------------------------------
(ns ep1.core
     (:gen-class)
     (:require [clojure.set :as set]))

;; Basix functions
-----------------------------------------------------------------------
-------
(defn CreateR [i_array j_array]
 (let [ result (map vector i_array j_array) ]
 (vec result))
)

(defn CreateReflectiveR[R PossibleReflectiveValues]
 (def DataRemoveCondition (fn [x] (contains? (set R) x)) )
 (let [return (remove DataRemoveCondition PossibleReflectiveValues)]
 (clojure.set/union R return)
 )
)

;; Transitive function using recursion
-----------------------------------------------------------

(defn CreateBinaryMatrix [n coords]
 (let [matrix (vec (repeat n (vec (repeat n 0))))] ; cria matriz
vazia
   (do
     (def result matrix)
     (doseq [[x y] coords]
       (do
         (def result (assoc-in result [(- x 1) (- y 1)] 1))
       )
```

```
    )
  )
  result
 )
)

(defn MatrixMult
 [mat1 mat2]

 (let [
    Transpose #(apply map vector %)
    DotProduct #(reduce + (map * %1 %2))
    row-mult (fn [mat row](map (partial DotProduct row)
                      (Transpose mat)))
    ]
  (do
    (def result (map (partial row-mult mat2)
      mat1
    ))
   (def result  (vec (map vec result)) )
  )
  result
 )
)

(defn TranformMatrixInBinaryMatrix [n matrix]
 (do
  (def result (vec (repeat n (vec (repeat n 0))))  )
  (def i 0)
  (def j 0)
  (while (< i n)
    (do
      (while (< j n)
        (do
          (if (> (get-in matrix [i j]) 0)
            (do
```

```
              (def result (assoc-in result [i j] 1))
            )
          )
          (def j (+ j 1))
        )
      )
      (def j 0)
      (def i (+ i 1))
    )
  )
 )
 result
)

(defn GetBinatyMatrixIndex[n matrix]
  (do
    (def result [])
    (def i 0)
    (def j 0)
    (while (< i n)
      (do
        (while (< j n)
          (do
            (if (= 1 (get-in matrix [i j]))
              (do
                (def result (conj result [(+ i 1) (+ j 1)]))
              )
            )
            (def j (+ j 1))
          )
        )
        (def j 0)
        (def i (+ i 1))
      )
    )
 )
```

```
   result
)

(defn MatrixOr [n matrix-a matrix-b]
   (do
     (def result (vec (repeat n (vec (repeat n 0)))) )
     (def i 0)
     (def j 0)
     (def matrix2 '[[0 0 1] [0 0 1] [0 0 1]])
     (while (< i n)
       (do
         (while (< j n)
           (do
             (if (or (= 1 (get-in matrix-a [i j])) (= 1 (get-in
matrix-b [i j])))
               (do
                 (def result (assoc-in result [i j] 1))
               )
             )
             (def j (+ j 1))
           )
         )
         (def j 0)
         (def i (+ i 1))
       )
     )
 )
   result
)

(defn GetTransitiveR [n R Rn]
 (def Rn_plus1 (TranformMatrixInBinaryMatrix n (MatrixMult R Rn)) )
 (if (= Rn_plus1 Rn)
   Rn
   (MatrixOr n Rn (GetTransitiveR n R Rn_plus1) )
 )
```

```
)

(defn Transitive [n R]
 (def R_binary (CreateBinaryMatrix n R))
 (def R_final_binary (GetTransitiveR n R_binary R_binary) )
 (def R_final (GetBinatyMatrixIndex n R_final_binary) )
 R_final
)

;; Main function
--------------------------------------------------------------------------------
(defn -main []

 (def A [1 2 3])
 (println "A:" A)

 (def n (count A))

 (def R
   (CreateR [1, 2, 3]
            [2, 3, 3])
 )
 (println "R:" R)

 (def PossibleReflectiveValues
   (CreateR A A)
 )

 (def ReflectiveR
   ( CreateReflectiveR R PossibleReflectiveValues )
 )
 (println "Fecho Reflexivo  de R | Reflective(R):" ReflectiveR)

 (def TransitiveR
   ( Transitive n R)
```

```
 )
 (println "Fecho Transitivo de R | Transitive(R):" TransitiveR)
)
```