

RELATÓRIO DE DESENVOLVIMENTO PRÁTICO -> TRABALHO I

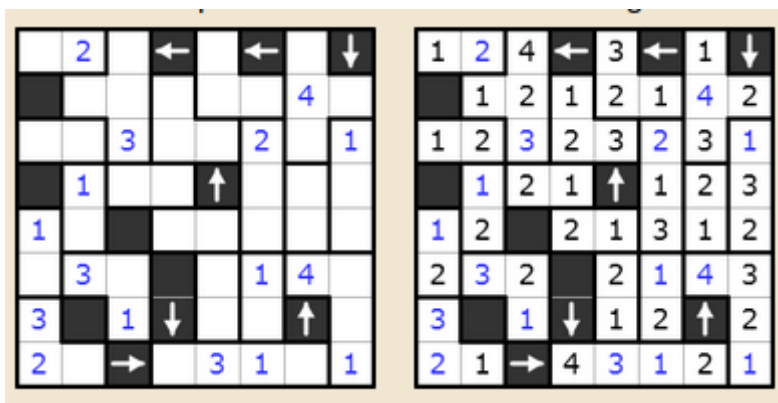
INE5416 - Paradigmas de Programação

Puzzle: Makaro (マカロ)

Alunos: Vitor Marzarotto, Raquel Behrens, Tainá da Cruz

1. Análise do Problema

Makaro (マカロ) é um quebra-cabeça lógico publicado pela primeira vez na revista japonesa de quebra-cabeças Nikoli.



Existem 3 regras para resolver esse puzzle:

1. Inserir um número em cada célula branca do diagrama para que cada região de N células contenha todos os números de 1~N exatamente uma vez.
2. Os mesmos números não devem ser ortogonalmente adjacentes.
3. Uma seta em uma célula preta aponta para a célula adjacente ortogonalmente com o número absolutamente mais alto.

2. Entrada do Usuário

Para resolver uma matriz makaro com o programa é preciso entender como a matriz que representa o tabuleiro é definida no código. Além disso, a entrada da matriz do puzzle é atualizada diretamente no código, por conta do problema ao fazer inputs, a ser comentado na seção 5 do relatório.

A matriz *makaro*, que é do tipo grid, está localizada no módulo *main*. Cada linha é composta por n células, que possuem uma tupla de tamanho 4. O primeiro valor da tupla diz respeito a posição da linha na matriz. O segundo valor é a posição da coluna na matriz.

O terceiro valor diz respeito a qual região aquela célula pertence. Se a célula for uma flecha ou um quadrado preto, a região DEVE ser 0.

E, por último, a quarta célula representa o valor em si dentro dela, Value, sendo esse Fixed, Possible, uma célula preta (Black) ou uma Arrow.

Se for uma célula onde deverá ser inserida uma flecha, Value deve ser Arrow, o valor que segue Arrow deve ser das flechas. Se a flecha for para a direita, o valor a ser colocado deve ser 1, se for para baixo, o valor deve ser 2, se for para a esquerda, o valor deve ser 3 e, se for para cima, o valor deve ser 4.

Se for uma célula que está vazia, Value deve ser Possible seguido de uma lista vazia [].

Se for uma célula com valor já inserido, Value deve ser Fixed e o número do valor.

Se for uma célula preta (onde não há valores e não se pode inserir valores), Value deve ser Black.

Para executar o programa, deve-se ter Haskell instalado no computador, compilar os arquivos com:

- `ghc --make Backtracking.hs CheckRegion.hs Conversors.hs main.hs Matriz.hs ParseNewMatrix.hs Prunning.hs Transform.hs Verification.hs -dynamic`

Executar com:

- `./main`

3. Estratégia e Solução Aplicada

A estratégia aplicada se baseou na técnica de programação de “tentativa e erro” (*backtracking*), utilizando a linguagem Haskell. A função **backtracking** (*Backtracking.hs*) do código funciona analisando o tabuleiro com 2 funções: **hasImpossible**, que retorna verdadeiro quando tabuleiro possui pelo menos 1 casa em que não há nenhum número que possa ser colocado sem quebrar as regras do jogo; e **hasPossible**, que retorna verdadeiro quando tabuleiro possui pelo menos 1 casa que pode receber mais que 1 número). Se nenhuma delas retornar verdadeiro, significa que o tabuleiro está resolvido, pois não há casas impossíveis e nem com possibilidades, ou seja, todas estão com um único número.

Backtracking recebe uma lista (que vai ser utilizada como pilha) de tabuleiros e retorna um tabuleiro. Ela aplica a função **while** sobre o tabuleiro no topo da Pilha, que, a partir das regras do jogo, vai retirando o máximo de possibilidades possível. Após isso, irá checar primeiramente se o tabuleiro fica impossível de ser resolvido (função **hasImpossible** retorna **True**). Nesse caso, ele retira a cabeça da pilha, e chama a si mesma sobre a cauda da pilha. Caso não seja impossível, irá verificar se ainda há possibilidades (função **hasPossible** retorna **True**), e, se houver, ele irá escolher uma das casas com menos possibilidades possível. Então, irá empilhar cópias do tabuleiro na pilha com a casa preenchida com cada possibilidade, e chamará recursivamente a pilha. Se não for possível nem impossível, o tabuleiro está resolvido, então a função backtracking apenas retorna a cabeça da pilha.

A função **while** (*Backtracking.hs*) chama a função **tiraPossibilidades** (*Backtracking.hs*). Essa função primeiramente chama a função **pruningCellPossibilities** (*Prunning.hs*), que checa os números fixos de cada região, e os retira da lista de possibilidades das células Possible daquela região. Após isso, chama a função **verifyOrthogonallyAdjacency** (*Verification.hs*), que poda as listas de possibilidades conforme as regras de adjacência do jogo. E, em seguida, chama a função **transformOnePossibilityLists** (*Transform.hs*), que tornam células Possible, as quais possuem uma lista com uma única possibilidade, em Fixed com a única possibilidade.

A respeito dos módulos, segue abaixo a descrição de cada um:

- *Backtracking* - Agrupa as funções que dizem respeito ao backtracking, a função principal que resolve o problema, a partir da retirada de possibilidades e testes dessas.

```
backtracking:: [Grid] -> Grid
backtracking [] = []
backtracking (a:b)
  | hasImpossible (while a) = backtracking b
  | hasPossible (while a) = backtracking ((generateGrids (while a)(mayToCell (getBestCell (while a) 2))) ++ b)
  | otherwise = (while a)
```

- *CheckRegion* - Retorna a quantidade de regiões de uma matriz, como uma função auxiliar para os outros módulos, a fim de que o código seja genérico e funcione com qualquer entrada.

```

module CheckRegion where
  import Matriz

  -- Ver se tem determinado número em uma região
  -- Função para checar quantas regiões a matriz tem

  convLineRegion :: Row -> Int
  convLineRegion [] = 0
  convLineRegion (x:[]) = getRegion x
  convLineRegion (x:xs) | ((getRegion x) > convLineRegion xs) = getRegion x
                        | otherwise = convLineRegion xs

  convMatrixRegion :: Grid -> Int
  convMatrixRegion [] = 0
  convMatrixRegion (x:[]) = convLineRegion x
  convMatrixRegion (x:xs) | (convLineRegion x > convMatrixRegion xs) = convLineRegion x
                        | otherwise = convMatrixRegion xs

  amountOfRegions :: Grid -> Int
  amountOfRegions x = convMatrixRegion x

```

- *Convertors* - Agrupa as funções de conversão de dados, para auxiliar na hora de mostrar valores na saída.

```

module Convertors where
  import Matriz

  -- Converte Matriz em formato String

  arrowString :: Value -> String
  arrowString x | ((getArrowValue x) == 1) = ">"
                | ((getArrowValue x) == 2) = "v"
                | ((getArrowValue x) == 3) = "<"
                | ((getArrowValue x) == 4) = "^"

  blackString :: Value -> String
  blackString x = "#"

  valueToChar :: Value -> String
  valueToChar x | (isArrow x == True) = (arrowString x)
                | (isBlack x == True) = (blackString x)
                | otherwise = show (getFixedValue(x))

  rowToString :: Row -> String
  rowToString [] = "\n"
  rowToString ((a,b,c,d):xs) = "[ "++(valueToChar d)++" ]" ++ rowToString xs

```

- *Main* - Módulo principal de execução do programa.

```

module Main (main) where
    import Matriz
    import Backtracking
    import Convertors
    import ParseNewMatrix

    -----
    -- MAKARO 2 --
    -----

    makaro :: Grid
    makaro =
        [
            [(1,1,0,Arrow 1), (1,2,2,Possible []), (1,3,2,Fixed 4), (1,4,0,Arrow 1),
             [(2,1,1,Possible []), (2,2,2,Possible []), (2,3,2,Possible []), (2,4,3,Possible []),
             [(3,1,1,Possible []), (3,2,0,Arrow 1), (3,3,5,Fixed 4), (3,4,5,Fixed 4),
             [(4,1,16,Fixed 2), (4,2,16,Possible []), (4,3,0,Arrow 4), (4,4,5,Possible []),
             [(5,1,0,Arrow 1), (5,2,7,Possible []), (5,3,7,Possible []), (5,4,8,Possible []),
             [(6,1,17,Possible []), (6,2,14,Possible []), (6,3,7,Possible []), (6,4,0,Arrow 1),
             [(7,1,0,Arrow 1), (7,2,14,Fixed 5), (7,3,13,Possible []), (7,4,13,Fixed 4),
             [(8,1,14,Fixed 3), (8,2,14,Possible []), (8,3,14,Fixed 4), (8,4,13,Possible [])]
        ]

    allToString :: [Grid] -> Int -> String
    allToString [] n = "lista vazia"
    allToString array 0 = gridToString (array !! 0)
    allToString array n = gridToString (array !! n) ++ "\n" ++ allToString array (n-1)

```

- *Matriz* - Agrupa todas as funções e valores referente a construção da matriz a ser analisada.

```

module Matriz where

data Value = Fixed Int | Possible [Int] | Arrow Int | Black deriving (Show, Eq)
type Cell = (Int,Int,Int,Value) -- (Linha, Coluna, Região, Valor)
type Row = [Cell] -- Arrow: 1 - Direita, 2 - Baixo, 3 - Esquerda, 4 - Cima
type Grid = [Row]

-----
-- FUNÇÕES --
-----

-- Precisa pegar o Value, mudar o Value, e retorna uma nova tupla com os valores atualizados
ehIgual :: Int -> [Int] -> Bool
ehIgual x fixedValues =
    if x `elem` fixedValues then
        False
    else
        True

filtrar :: (Int -> [Int] -> Bool) -> [Int] -> [Int] -> [Int]
filtrar funcao lista fixedValues = [a | a <- lista, funcao a fixedValues]

-----
-- GETTERS --
-----

getRow :: (Int,Int,Int,Value) -> Int
getRow (a,_,_,_) = a

getCol :: (Int,Int,Int,Value) -> Int
getCol (_,b,_,_) = b

```

- *Prunning* - Retira das listas de possibilidades de cada célula, agrupando em regiões, os números de cada região que são Fixed (para todas as regiões);

```

37 -- retorna nova matriz com a lista de possibilidades atualizadas para uma região
38 pruningCellPossibilities :: Grid -> Int -> Grid
39 pruningCellPossibilities grid 1 = (deleteFixedValuesOfRegions grid 1 (getFixedValuesOfMatrix grid 1))
40 pruningCellPossibilities grid amountRegions =
41   pruningCellPossibilities (deleteFixedValuesOfRegions grid amountRegions (getFixedValuesOfMatrix grid amountRegions)) (amountRegions-1)
42

```

- *Transform* - Checa célula por célula, e, se uma célula tiver o Value como Possible e sua lista de possibilidades com apenas um número, transforma o Value da célula em Fixed com esse número.

```

32 -- checa, para todas as células, se as listas de possibilidades possuem apenas uma possibilidade, para torná-la Fixed
33 transformOnePossibilityLists :: Grid -> Grid
34 transformOnePossibilityLists grid = transformMatrix grid
35

```

- *Verification* - Varre a matriz e tira da lista de possibilidades das células, com Value Possible, os números que quebram as regras do jogo.

```

184
185 -- função que verifica as regras de adjacência do jogo para cada célula
186 -- retorna a grid com listas de possibilidades menores, se alguma possibilidade não estiver de acordo com as regras do jogo
187 verifyOrthogonallyAdjacency :: Grid -> Grid
188 verifyOrthogonallyAdjacency grid = verifyMatrix grid grid

```

- *ParseNewMatrix* - Seta as listas de possibilidades das células que não têm valor definido.

```

46 -- Função para setar as listas de possibilidades para todas as células
47 -- manda o retorno da função getListMaxPossibilidadesPorRegiao para setListMatrix
48 setListsOfPossibilities :: Grid -> Grid
49 setListsOfPossibilities grid = setListMatrix grid (getListMaxPossibilidadesPorRegiao grid (amountOfRegions grid))
50

```

A saída do programa, ao executar as funções para resolver o problema, é a seguinte:

```
~/Área de trabalho/Paradigmas/makaro main * ghc --m
ation.hs -dynamic
[8 of 8] Compiling Main          ( main.hs, main.o )
Linking main ...
~/Área de trabalho/Paradigmas/makaro main * ./main
-----
|  MAKARO SOLVER  |
-----

[ > ][ 3 ][ 4 ][ < ][ 2 ][ 3 ][ 4 ][ 1 ]
[ 2 ][ 1 ][ 2 ][ 3 ][ 1 ][ 2 ][ 1 ][ v ]
[ 1 ][ > ][ 4 ][ 2 ][ # ][ 3 ][ 2 ][ 4 ]
[ 2 ][ 1 ][ ^ ][ 1 ][ 3 ][ 1 ][ > ][ 5 ]
[ > ][ 3 ][ 2 ][ 3 ][ 4 ][ 3 ][ 2 ][ 1 ]
[ 1 ][ 2 ][ 1 ][ ^ ][ 2 ][ 1 ][ v ][ 2 ]
[ > ][ 5 ][ 3 ][ 2 ][ 1 ][ > ][ 3 ][ 4 ]
[ 3 ][ 1 ][ 4 ][ 1 ][ 2 ][ 1 ][ 2 ][ ^ ]
```

As setas são representadas pelas string 'v', '>', '<' e '^'. As células totalmente pretas são representadas pela string '#'. Não foi possível realizar a definição das regiões no output, então as mesmas só são conhecidas dentro das 4-tuplas das células.

4. Organização do Grupo

A comunicação do grupo ocorreu por meio de um grupo no aplicativo Telegram. Abaixo, descrição das atividades dos membros:

Vitor: Funções do arquivo de Backtracking que resolvem o problema a partir das funções do arquivo Prunning. Algumas funções para printar a matriz.

Raquel: Funções do arquivo Prunning; Funções do arquivo CheckRegion; Funções do arquivo Transform; Funções do arquivo Verification; Funções do arquivo ParseNewMatrix; Algumas funções do arquivo Matriz.

Tainá: Definição da estrutura da matriz baseada em valores *Fixed* ou *Possible*. Algumas funções para pegar dados e de retorno booleano. Separação do arquivo

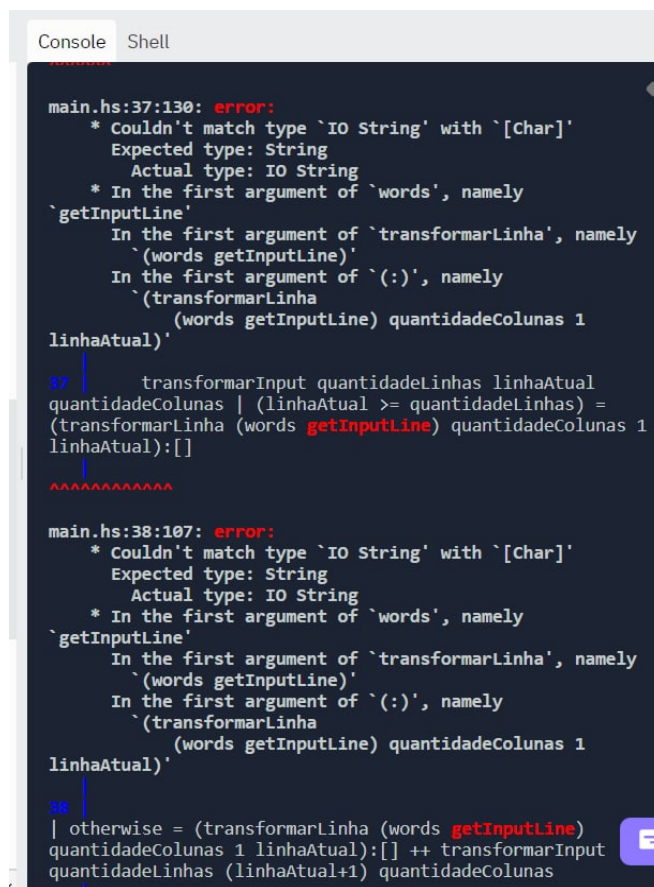
principal em vários módulos. Algumas funções de conversão de dados e para imprimir a matriz (em `Conversors.hs`). Relatório.

5. Dificuldades Encontradas

O grupo encontrou algumas dificuldades na resolução do problema. Um ponto a se destacar diz respeito à sintaxe do Haskell. Por ser um paradigma novo, o grupo precisou se adaptar às novas possibilidades e bloqueios que a linguagem proporciona.

Outra dificuldade resultou da tentativa de inserir um input para a matriz, fazendo com que o próprio programa construísse a sua matriz do jogo Makaro. O grupo teve dificuldades ao tentar utilizar as funções de input nativas do Haskell (`getLine`) para pegar linha por linha da inserção. Por conta disso, o input não foi possível, então a matriz é setada dentro do próprio sistema, no arquivo `main.hs`.

Print do erro:



```
main.hs:37:130: error:
    * Couldn't match type `IO String' with `[Char]'
      Expected type: String
      Actual type: IO String
    * In the first argument of `words', namely
      `getInputLine'
      In the first argument of `transformarLinha', namely
      `(words getInputLine)'
      In the first argument of `(:)', namely
      `(transformarLinha
        (words getInputLine) quantidadeColunas 1
        linhaAtual)'
37 |     transformarInput quantidadeLinhas linhaAtual
   |     quantidadeColunas | (linhaAtual >= quantidadeLinhas) =
   |     (transformarLinha (words getInputLine) quantidadeColunas 1
   |     linhaAtual):[]
   |
   |
main.hs:38:107: error:
    * Couldn't match type `IO String' with `[Char]'
      Expected type: String
      Actual type: IO String
    * In the first argument of `words', namely
      `getInputLine'
      In the first argument of `transformarLinha', namely
      `(words getInputLine)'
      In the first argument of `(:)', namely
      `(transformarLinha
        (words getInputLine) quantidadeColunas 1
        linhaAtual)'
38 | | otherwise = (transformarLinha (words getInputLine)
   | | quantidadeColunas 1 linhaAtual):[] ++ transformarInput
   | | quantidadeLinhas (linhaAtual+1) quantidadeColunas
```


Também não foi possível, como comentado anteriormente, definir graficamente as regiões da matriz no output. Como as regiões no Makaro são aleatórias, a depender do tamanho do tabuleiro, não foi possível arquitetar uma solução gráfica para isso.