UNIVERSIDADE FEDERAL DE SANTA CATARINA
CAMPUS FLORIANÓPOLIS
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
CURSO CIÊNCIAS DA COMPUTAÇÃO

INE5416 – PARADIGMAS DE PROGRAMAÇÃO

PEDRO AUGUSTO DA FONTOURA (22215098)
ERIK ORSOLIN DE PAULA (22102195)
RAFAEL CORREA BITENCOURT (22203673)

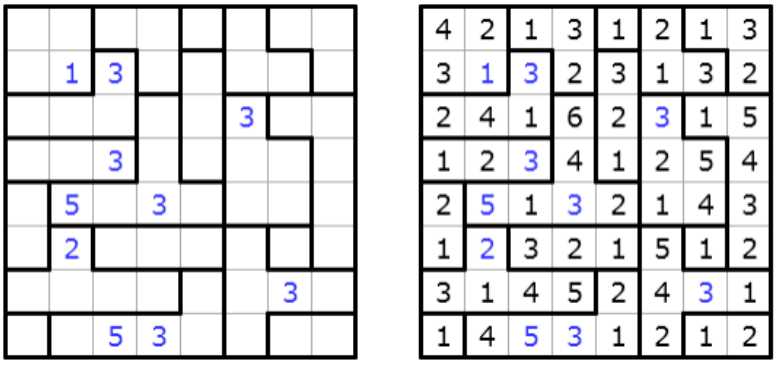
TRABALHO 1 DE PARADIGMAS DE PROGRAMAÇÃO

FLORIANÓPOLIS

2024

KOJUN

O jogo consiste em preencher todas as regiões de um tabuleiro, que pode variar de 6x6 até 17x17, com números de 1 a n, sem repetição, onde n é o tamanho da região a ser preenchida. Números adjacentes não podem ser iguais e, se estiverem na mesma região, o número diretamente acima deve ser maior do que o abaixo dele.



RESUMO DO ALGORITMO

O algoritmo em questão emprega a técnica de retrocesso (backtracking), na qual são atribuídos valores provisórios a cada casa e, subsequentemente, verifica-se se as regras estabelecidas foram violadas. Para tal propósito, foram utilizadas as funções posicaoValida e resolver. A primeira função verifica se o posicionamento do número k (passado como parâmetro pela função resolver) está em conformidade com as regras do jogo, enquanto a segunda função percorre todo o tabuleiro, tentando posicionar números e validando-os através da primeira função.

No que diz respeito ao armazenamento dos valores e das regiões, foram utilizadas duas matrizes (listas de listas de inteiros) denominadas tabuleiro e regioes. A matriz tabuleiro contém os valores que são efetivamente posicionados no tabuleiro, e a matriz regioes possui números que se repetem para representar uma mesma região.

Para posicionar os elementos na matriz, considerando que o Haskell utiliza constantes para armazenar os seus valores, foi necessário utilizar uma função auxiliar denominada substituir2D. Esta função faz uso da função pronta para dividir a matriz em partes antes e depois do valor a ser substituído, removendo o valor antigo e criando uma nova matriz com o novo valor concatenado na posição.

```haskell
-- Define o módulo Matriz que exporta a função substitui2D.
module Matriz (substitui2D) where

-- Define a função substitui2D que substitui um elemento em uma posição específica do tabuleiro.
substitui2D :: [[Int]] -> Int -> Int -> Int -> [[Int]]
substitui2D tabuleiro linha coluna k =
    let (antes, (x:depois)) = splitAt linha tabuleiro  -- Divide o tabuleiro em duas partes na linha especificada.
        (antesColuna, (_:depoisColuna)) = splitAt coluna x  -- Divide a linha especificada na coluna especificada.
    in antes ++ [antesColuna ++ [k] ++ depoisColuna] ++ depois  -- Constrói um novo tabuleiro com o elemento substituído.
```

```haskell
-- Define o módulo Validacao que exporta a função ehPosicaoValida.
module Validacao (ehPosicaoValida) where

-- Define a função ehPosicaoValida que verifica se um número k pode ser colocado em uma posição específica do tabuleiro.
ehPosicaoValida :: [[Int]] -> [[Int]] -> Int -> Int -> Int -> Bool
ehPosicaoValida tabuleiro regioes linha coluna k =
    let regiao = regioes !! linha !! coluna  -- Obtém a região da posição atual.
        -- Obtém todos os valores na mesma região que a posição atual.
        valoresRegiao = [tabuleiro !! i !! j | i <- [0..length tabuleiro - 1], j <- [0..length tabuleiro - 1], regioes !! i !! j == regiao]
        -- Define as posições dos vizinhos diretos.
        vizinhos = [(linha - 1, coluna), (linha + 1, coluna), (linha, coluna - 1), (linha, coluna + 1)]
        -- Obtém os valores dos vizinhos diretos que estão dentro dos limites do tabuleiro.
        valoresVizinhos = [tabuleiro !! i !! j | (i, j) <- vizinhos, i >= 0, i < length tabuleiro, j >= 0, j < length tabuleiro]
    in
        -- Verifica se k não está presente na região ou entre os vizinhos e cumpre as condições adicionais para as bordas do tabuleiro.
        k `notElem` valoresRegiao &&
        k <= length valoresRegiao &&
        k `notElem` valoresVizinhos &&
        (linha == 0 || regioes !! (linha - 1) !! coluna /= regiao || tabuleiro !! (linha - 1) !! coluna >= k) &&
        (linha == length tabuleiro - 1 || regioes !! (linha + 1) !! coluna /= regiao || tabuleiro !! (linha + 1) !! coluna <= k)
```

INSTRUÇÕES DE USO

Os dados serão fornecidos por meio das variáveis tabuleiro e regioes dentro da função main, utilizando uma matriz quadrada. Os resultados serão exibidos no terminal. Para executar o programa, siga estes passos no terminal (a partir do diretório que contém o arquivo main.hs):

1. ghc main
2. ./main

Tabuleiro a ser preenchido para a solução:

```haskell
main = do
    let tabuleiro =    [[0,0,0,0,0,0,0,0],
                        [0,1,3,0,0,0,0,0],
                        [0,0,0,0,0,3,0,0],
                        [0,0,3,0,0,0,0,0],
                        [0,5,0,3,0,0,0,0],
                        [0,2,0,0,0,0,0,0],
                        [0,0,0,0,0,0,3,0],
                        [0,0,5,3,0,0,0,0]]

        regioes =       [[1,1,2,2,3,4,5,5],
                        [1,1,6,2,7,4,4,5],
                        [6,6,6,8,7,9,10,10],
                        [11,11,11,8,7,9,9,10],
                        [12,8,8,8,8,9,9,10],
                        [12,13,14,14,14,15,16,10],
                        [13,13,13,13,17,15,15,15],
                        [18,17,17,17,17,15,19,19]]
```

Foi utilizado a matriz exemplo do site fornecido pelo professor.

ORGANIZAÇÃO DO GRUPO

O grupo se reuniu para aplicar os conhecimentos adquiridos na matéria de Paradigmas de Programação. Após discussões e estudos, desenvolvemos o trabalho de forma colaborativa, utilizando ideias conjuntas para encontrar soluções eficazes.

DIFICULDADES ENCONTRADAS

O grupo enfrentou dificuldades principalmente no entendimento da programação funcional e no desenvolvimento do problema em Haskell, uma linguagem nova para nós. Para superar esse desafio, primeiro desenvolvemos um código em Python, o que nos ajudou a compreender melhor o problema. Em seguida, focamos apenas na sintaxe de Haskell, facilitando a adaptação e a resolução do problema na nova linguagem.