Universidade Federal de Santa Catarina
Campus Trindade
Centro Tecnológico
INE5416 - Paradigmas de Programação

Relatório Trabalho 1
Programação Funcional - Haskell

Leonardo Brito (21200896)
Fernanda Müller (21202109)
Isac Martins (21203361)

Florianópolis, 30 de setembro de 2023

1. Análise do problema

Kojun é um jogo de lógica semelhante ao sudoku. O jogador recebe um tabuleiro parcialmente preenchido e com regiões demarcadas e o seu objetivo é preencher essas regiões seguindo um conjunto simples de regras. Dentre as principais estão: O jogador não pode repetir um mesmo número dentro de uma região, o jogador não pode colocar numa posição um número que já está em uma posição imediatamente adjacente (cima, baixo, direita e esquerda) e, numa linha vertical dentro de uma mesma região, as células devem estar posicionadas de forma crescente de baixo para cima.

2. Solução Adotada

Temos dois tabuleiros, o primeiro contendo as posições e os valores de cada posição (0 se vazia) e o segundo demarcando as regiões de 1 à n, da esquerda para direita e de cima para baixo.

```haskell
-- Banco de Tabuleiros
-- Tabuleiros retirados do site: https://www.janko.at/Raetsel/Kojun/index.htm
-- Tabuleiros 3
get_valores_tabuleiro6x6 :: Tabuleiro
get_valores_tabuleiro6x6  = [[0, 0, 0, 0, 0, 2],
                             [2, 0, 0, 5, 0, 0],
                             [0, 0, 3, 0, 0, 4],
                             [0, 0, 0, 3, 0, 1],
                             [0, 0, 0, 0, 0, 0],
                             [0, 0, 3, 0, 2, 5]]

get_regioes_tabuleiro6x6 :: Tabuleiro
get_regioes_tabuleiro6x6 = [[0, 1, 2, 2, 3, 3],
                            [0, 1, 4, 3, 3, 3],
                            [0, 0, 4, 4, 4, 5],
                            [6, 6, 7, 5, 5, 5],
                            [6, 6, 7, 8, 9, 9],
                            [10, 10, 8, 8, 8, 8]]
```

Inicialmente, o código mapeia o tabuleiro das regiões para conseguir um tabuleiro mapeado que separa a linha e coluna de cada elemento de uma região em uma tupla e agrupando em uma lista, fazendo isso para todas as regiões do tabuleiro e agrupando em uma lista de listas. Depois ele tenta resolver o tabuleiro.

```haskell
-- Recebe o tabuleiro que deve ser resolvido e seu tamanho
-- Mapeia o tabuleiro e tenta resolver ele
-- Retorna o tabuleiro resolvido
kojun :: Tabuleiro -> Tabuleiro -> Int -> IO String
kojun valoresTabuleiro regioesTabuleiro tamanho =
    let regioesMapeadas = mapearTabuleiro regioesTabuleiro tamanho
        tabuleiroResolvido = resolverTabuleiro 0 0 tamanho valoresTabuleiro regioesTabuleiro regioesMapeadas
    in return (formatarResultado tabuleiroResolvido)
```

Para resolver uma dada instância de um tabuleiro Kojun utilizamos um algoritmo de backtracking, onde várias soluções possíveis serão tentadas até uma delas dar certo ou o tabuleiro é marcado como não resolvível (Vazio).

Algoritmo: Para cada linha do tabuleiro e para cada posição executamos o algoritmo de backtracking. Primeiro verificamos se um número é possível naquela posição, se não tentamos outro, se sim retornamos o tabuleiro atualizado e tentamos ocupar a próxima região. Se em algum momento todos os valores possíveis para uma posição derem errado, voltamos para a última chamada e tentamos todas as outras soluções possíveis.

Função responsável por executar o algoritmo de backtracking em cada linha e coluna do tabuleiro:

```haskell
-- Loop principal que deve resolver o Tabuleiro varrendo posição por posição
-- Retorna o Tabuleiro original solucionado
resolverTabuleiro :: Int -> Int -> Int -> Tabuleiro -> Tabuleiro -> TabuleiroMapeado -> Tabuleiro
resolverTabuleiro i j tamanho valoresTabuleiro regioesTabuleiro regioesMapeadas
    -- Caso tenha percorrido toda matriz e solucionado o problema
    -- Retorna o tabuleiro solucionado
    | (i == tamanho - 1) && (j == tamanho) = valoresTabuleiro

    -- Varreu a linha até o último elemento
    -- Pula para a próxima linha e reinicia o processo
    | j == tamanho =
        resolverTabuleiro (i+1) 0 tamanho valoresTabuleiro regioesTabuleiro regioesMapeadas

    -- Verifica se a posição avaliada já está ocupada
    -- Caso sim, pula para a próxima da linha
    | (valoresTabuleiro !! i !! j) > 0 =
        resolverTabuleiro i (j+1) tamanho valoresTabuleiro regioesTabuleiro regioesMapeadas

    -- Posição não está ocupada
    -- Procura um numero para ocupá-la
    | otherwise =
        -- Procura a partir do maior número da regiao da posição (Tamanho da Região)
        let valMax = tamanhoRegiao regioesMapeadas (regioesTabuleiro !! i !! j)
        in ocuparPosicao valMax i j tamanho valoresTabuleiro regioesTabuleiro regioesMapeadas
```

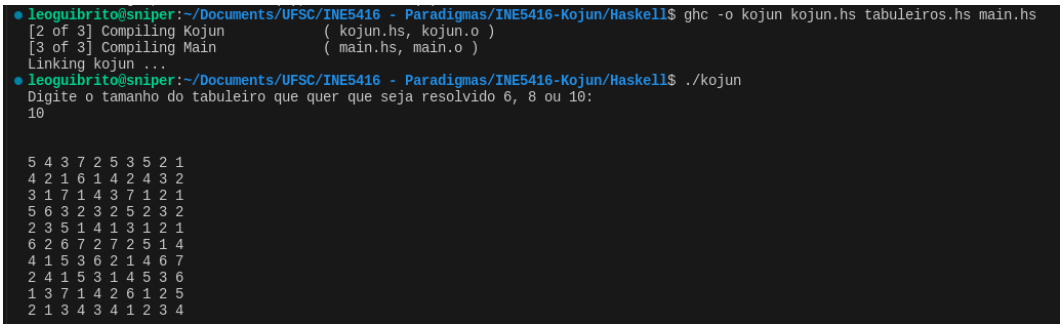
Função responsável por efetivamente aplicar o algoritmo de backtracking:

```haskell
-- Usa o algoritmo de Backtracking para tentar ocupar a região com cada número possível
-- Se nenhum número for válido significa que a posição não é ocupável e o tabuleiro não é resolvível
-- Retorna um Tabuleiro Vazio ou Resolvido como resposta
ocuparPosicao :: Valor -> Int -> Int -> Int -> Tabuleiro -> Tabuleiro -> TabuleiroMapeado -> Tabuleiro
ocuparPosicao valorPosicao i j tamanho valoresTabuleiro regioesTabuleiro regioesMapeadas
    -- Se não foi possível ocupar a posição não é possível resolver o tabuleiro
    -- Retorna um tabuleiro vazio
    | valorPosicao <= 0 = [[]]

    -- Verifica se é possivel inserir aquele valor na posição
    | otherwise =
        -- Se possível atualiza o tabuleiro e chama a função Kojun na próxima posição
        if valorPossivel valorPosicao i j tamanho valoresTabuleiro regioesTabuleiro regioesMapeadas then
            let tabuleiroAtualizado = atualizarTabuleiro valorPosicao i j valoresTabuleiro
                tabuleiro = resolverTabuleiro i (j + 1) tamanho tabuleiroAtualizado regioesTabuleiro regioesMapeadas

                -- Caso as próximas iterações forem bem sucedidas retorna o tabuleiro resolvido até o momento
            in  if not (null tabuleiro || all null tabuleiro) then
                    tabuleiro
                -- Se alguma der errado tenta resolver mais uma vez essa posição com um número menor
                else
                    ocuparPosicao (valorPosicao - 1) i j tamanho valoresTabuleiro regioesTabuleiro regioesMapeadas

        -- Se não chama a função mais uma vez com um valor de posição menor
        else
            ocuparPosicao (valorPosicao - 1) i j tamanho valoresTabuleiro regioesTabuleiro regioesMapeadas
```

Note que para cada posição ele verifica se o número tentado é válido aplicando as 3 regras mencionadas anteriormente (Abstraídas em outras funções).

Por fim, após a execução do algoritmo a função que recebeu o tabuleiro original vai pegar o tabuleiro resolvido e formatar ele em formato de string para a função main imprimir, com o resultado final ficando assim:

Exemplo Tabuleiro 10x10:



3. Organização

O trabalho foi realizado ao longo das semanas disponíveis através de reuniões do discord onde nos juntamos para discutir nossas ideias e trabalhar no projeto utilizando a ferramenta de Live Share do Visual Studio Code.

4. Dificuldades Encontradas

A maior dificuldade que encontramos foi na tentativa de manusear e fazer modificações na matriz do tabuleiro em haskell, onde python podíamos usar loops for e o acesso a uma posição específica da matriz é uma tarefa bem simples, em haskell é mais complicado e requer a necessidade da implementação de funções recursivas e do uso métodos já disponíveis da linguagem, mas que foram complicados de implementar em um primeiro momento.