

# Relatório de implementação de resolvidor de puzzle Kojun

Gabriel de Vargas Coelho, 20200400

Florianópolis, 2022

## Análise do Problema

O jogo escolhido para esse trabalho foi o kojun. A escolha se deu simplesmente por gosto pessoal, uma vez que não gosto de jogos no estilo sudoku, o kojun foi, dentre as opções, o mais divertido.

O kojun se trata de um jogo simples, em que o objetivo é conseguir completar um tabuleiro com números, respeitando 3 regras:

1. Casas adjacentes ortogonalmente devem possuir números distintos;
2. O número máximo para uma casa é determinado pelo número de casas do grupo, e dentro de um grupo não pode haver repetição de números;
3. Casas de um mesmo grupo e de mesma coluna, devem apresentar numeração decrescente no sentido de cima para baixo.

Assim, o desafio para a implementação de uma solução ao kojun se dá pelo fato de haverem muitas “respostas” corretas para um tabuleiro. Uma vez traçada a estratégia para o preenchimento das casas do tabuleiro, o segundo desafio é a verificação da corretude da solução. Na próxima seção são explicitados ambas as estratégias.

## Solução Proposta

A primeira coisa a ser destacada aqui é a modelagem do tabuleiro. Optou-se por utilizar a estrutura de tupla para modelar uma casa do tabuleiro, da seguinte forma:  $([Int], Int)$ , em que o primeiro valor é uma lista de todos os valores possíveis para aquela casa, e o segundo valor é um indicador do grupo ao qual aquela casa pertence. Uma vez definida a modelagem de uma casa do tabuleiro, o tabuleiro em si é somente uma matriz, em que cada posição é do tipo  $([Int], Int)$ . Para sinalizar uma casa vazia é usado uma lista com o valor 0.

```
[ ([0],0), ([0],1), ([4],1), ([0],1), ([2],3), ([0],4),  
  ([0],0), ([0],2), ([3],1), ([0],3), ([0],3), ([0],3),  
  ([1],0), ([4],0), ([0],5), ([4],3), ([0],10), ([0],10),  
  ([0],6), ([5],7), ([0],5), ([0],9), ([0],9), ([2],10),  
  ([0],6), ([0],7), ([0],7), ([0],8), ([3],8), ([0],10),  
  ([6],7), ([2],7), ([0],7), ([2],8), ([0],8), ([5],8)]
```

Imagem 1: Tabuleiro 6x6 usando a modelagem proposta

		4		2	
		3			
1	4		4		
	5				2
				3	
6	2		2		5

Imagem 2: Tabuleiro modelado pela imagem 1. Fonte: [1]

A primeira coisa feita pelo algoritmo de resolução é definir os valores possíveis para cada casa em branco. Por exemplo, seguindo a Imagem 2, a primeira casa do tabuleiro (casa superior esquerda), tem como opções os valores 2 e 3, assim em nossa modelagem seria o seguinte:  $([2,3],0)$ . Após isso é feita a geração de todos os tabuleiros possíveis a partir da escolha dos valores disponíveis na primeira casa em branco disponível. Em nosso exemplo, seriam gerados dois tabuleiros, um em que a primeira casa teria o valor  $([2],0)$  e outro tabuleiro que teria o valor  $([3],0)$  na primeira casa. Essa geração é feita pela função abaixo:

```
-- Gera novos tabuleiros com base na primeira casa do tabuleiro que possui múltiplas possibilidades
-- Para cada possibilidade um novo tabuleiro é gerado
generateBoards :: Board -> [Board]
generateBoards board =
  [b | boardAux <- [rows1 ++ [row1 ++ [c] : row2] ++ rows2 | c <- cs],
    let b = zipWith zip boardAux (map (map snd) board)]
  where
    (rows1, row:rows2) = break (any (not . single)) (map (map fst) board)
    (row1, cs:row2) = break (not . single) row
```

Imagem 3: Algoritmo para geração de tabuleiros com base na primeira casa encontrada que possua múltiplos valores possíveis. Adaptado de [2]

O algoritmo acima utiliza algumas funções da biblioteca Data.List do haskell, a fim de tornar o processo de desenvolvimento mais ágil.

Após a escolha feita, o primeiro tabuleiro gerado vai passar por um processo de redução de escolhas. Nesse processo, ao identificar uma casa do tabuleiro que possui múltiplas escolhas, é obtido os valores já configurados para o grupo ao qual essa casa pertence, e é feita a subtração desses valores no leque de escolhas da casa. Exemplificando o processo: considere que existem as seguintes casas no tabuleiro  $([1,3,4],1)$  e  $([4],1)$ , após passar pela redução teremos o seguinte:  $([1,3],1)$  e  $([4],1)$ .

O tabuleiro então segue para a avaliação, se o tabuleiro tiver todas as casas configuradas em um só valor, e cumprir as regras do jogo, explicitadas na seção anterior,

então esse tabuleiro é a solução do problema. Caso contrário, esse tabuleiro volta ao passo em que são gerados novos tabuleiros. Caso a árvore gerada para uma escolha possível não apresente resultado válido, é passado todo o procedimento sob a ótica de outra escolha, em caso de não haver solução é retornado ao usuário um array vazio.

No caso de sucesso, o usuário visualiza em seu terminal de comandos o tabuleiro, modelado com nossa proposta, com os valores da solução.

```
"Solucao do tabuleiro 6x6"  
[[[3],0),([2],1),([4],1),([1],1),([2],3),([1],4)]  
[[2],0),([1],2),([3],1),([5],3),([1],3),([3],3)]  
[[1],0),([4],0),([2],5),([4],3),([3],10),([4],10)]  
[[2],6),([5],7),([1],5),([2],9),([1],9),([2],10)]  
[[1],6),([4],7),([3],7),([4],8),([3],8),([1],10)]  
[[6],7),([2],7),([1],7),([2],8),([1],8),([5],8)]
```

Imagem 4: Solução possível do tabuleiro das Imagens 1 e 2

## Dificuldades e soluções

Uma dificuldade que ficou clara durante o desenvolvimento foi a pouca experiência com o paradigma funcional. Um exemplo disso são as funções que validam uma possível solução do tabuleiro, em que muitas vezes uma estrutura de for facilitaria muito, uma vez que, por costume, é mais simples acessar elementos de uma matriz utilizando índices. Contudo esse problema pôde ser resolvido estudando muito o exemplo disponibilizado em [2], que é uma implementação de um resolvedor de sudoku, ao qual muitas ideias foram adaptadas, e até mesmo é usado a mesma estratégia de solução do *puzzle*.

Além disso, outro problema foram as mensagens de erro, por se tratar de um problema mais complexo, por vezes me deparei com problemas de tipagem por erros bobos, como exemplo: no algoritmo da Imagem 3, a atribuição feita a b era inicialmente usado a seguinte sintaxe: `b <- ...`, porém isso faz com que somente o primeiro elemento de uma lista seja atribuído a b, esse erro de uso incorreto da sintaxe retornava um erro de tipagem no compilador, que demorou-se muito tempo, e muita paciência para ser devidamente corrigido.

## Referências

[1] JANKO.AT. **Kojun**. Disponível em: <<https://www.janko.at/Raetsel/Kojun/index.htm>>. Acesso em: 08 out. 2022.

[2] HUTTON, Graham. **Sudoku in Haskell**. 2022. Disponível em: <<http://www.cs.nott.ac.uk/~pszgmh/sudoku.lhs>>. Acesso em: 05 out. 2022.