

Universidade Federal de Santa Catarina - UFSC

Campus Florianópolis

Pedro Philippi Araujo

Trabalho 1

**Relatório Sobre o Desenvolvimento de um Resolvedor do Jogo
“Kojun”**

FLORIANÓPOLIS, 2023

Pedro Philippi Araujo

Trabalho 1

Relatório Sobre o Desenvolvimento de um Resolvedor do Jogo “Kojun”

Trabalho desenvolvido como requisito parcial
para obtenção de aprovação na disciplina de
PARADIGMAS DE PROGRAMAÇÃO da 4ª
fase no Curso de Ciências da Computação, na
Universidade Federal de Santa Catarina.

Prof. Maicon Rafael Zaiton

FLORIANÓPOLIS, 2023

1. INTRODUÇÃO:

O presente documento tem como objetivo relatar o processo de desenvolvimento de um código/ algoritmo de resolução do jogo “Kojun”, as dificuldades encontradas e os objetivos alcançados.

2. Kojun:

O Kojun possui regras similares ao jogo sudoku. Um quadrado composto de espaços de dimensões equivalentes é apresentado ao jogador e o mesmo deve preenchê-lo com números iguais seguindo as seguintes regras:

- 1- Cada região de tamanho N deve conter 1 até N números diferentes.
- 2- Números ortogonalmente adjacentes devem ser diferentes.
- 3- Caso dois números estejam na mesma região, o número superior deve ser maior ao seu inferior.

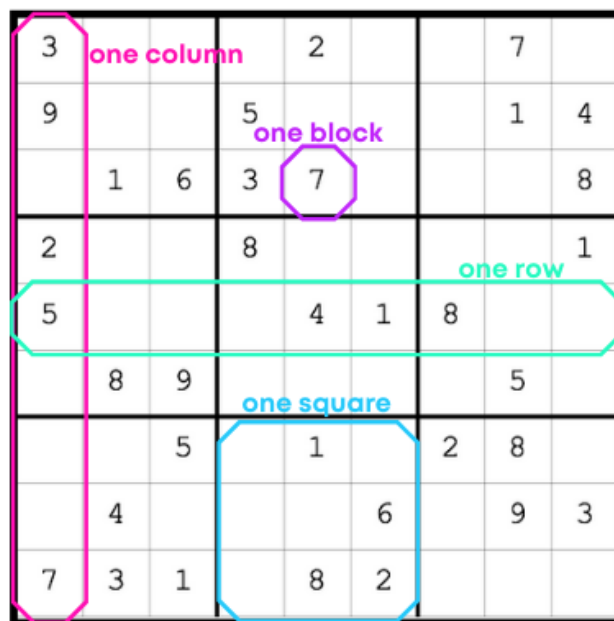


Figura 1: Exemplo de Sudoku Padrão, [link](#)

+		5			1	2		5	6
									2
		2	4						
4			5				3		
	4	6			5				
6			3	7	3		5	6	
				4			1		
			4		5	1	2	4	
5				2		2	5		6
	2		2			1			4

Figura 2: Exemplo de Sudoku Kojun.

3. IMPLEMENTAÇÃO:

3.1. Análise de Requisitos:

Após a análise do enunciado, foi identificado que os seguintes componentes seriam necessários para a resolução do problema:

- A Entrada, separadas em uma matriz com os números pré adicionados e uma matriz com as regiões das células.
- Matriz Grid para representar o tabuleiro do jogo, essa mesma foi feita com uma lista de listas.
- Matriz Regiões para facilmente mapear as regiões conexas.
- Matriz inicial para pular a verificação de células já pré-preenchidas.
- Uma Lista que enumera o número de regiões e o “tamanho” de cada Região.
- Função de resolução, que irá. testar todo o número possível em um tabuleiro se utilizando de recursividade

3.2. Algoritmo:

Após a análise de requisitos, o algoritmo escolhido foi dado por uma sugestão do professor em sala de utilizar backtracking para testar todas as possibilidades de um dado tabuleiro utilizando de recursividade. O algoritmo é dado em Pseudocódigo:

Faça Resolver em célula (X,Y):

Itera recursivamente sobre um I menor que o tamanho da região:

Se célula (X,Y) = I é válida [diferente de adjacente e maior que baixo na mesma região]:

Faça resolver em célula(X,Y+1)

Senão:

faça resolver célula(x,y) e I+1

3.3. Implementação em Python:

Com o Algoritmo pronto, primeiro uma implementação teste do código foi feita em Python, devido a familiaridade do autor com a linguagem. E apesar do uso abundante de funções específicas de lista do Python, o código foi bem sucedido em resolver Sudokus de até 10x10, porém com considerável tempo de execução, devido ao algoritmo testar TODAS as possibilidades de um tabuleiro (um Tabuleiro grande pode chegar na casa de trilhões de possibilidades).

```
def testeuno():
    for i in range(0, len(tamanhoregiones)):
        if tamanhoregiones[i] == 1:
            for j in range(0, len(gridregiao)):
                for k in range(0, len(gridregiao)):
                    if gridregiao[j][k] == i+1:
                        grid[j][k] = 1
                        gridinit[j][k] = 1

def choice(linha,coluna):
    if gridinit[linha][coluna] == 0:
        regioacel = gridregiao[linha][coluna]
        for i in range(1, tamanhoregiones[regioacel-1]+1):
            if ((coluna == 0) or (i != grid[linha][coluna-1])) and ((coluna == tamanhogrid-1 or i != grid[linha][coluna+1]
            if (linha == 0) or (regioacel == gridregiao[linha-1][coluna] and i < grid[linha-1][coluna]) or (regioacel
            if (linha == tamanhogrid-1) or (regioacel == gridregiao[linha+1][coluna] and i > grid[linha+1][coluna]

                grid[linha][coluna] = i

            if linha == tamanhogrid-1 and coluna == tamanhogrid-1:
                return 0
            elif linha != tamanhogrid-1 and coluna == tamanhogrid-1:
                choice(linha+1, 0)
            else:
                choice(linha,coluna+1)

    else:
        choice(linha,coluna+1)

    return 0

grid = [[2,0,0,0,1,0],[0,0,0,3,0,0],[0,3,0,0,5,3],[0,0,0,0,0,0],ca,[0,0,0,0,0,0]]
gridregiao = [[1,1,2,2,2,3],[4,4,4,4,4,3],[5,6,6,6,4,7],[5,5,5,6,7,7],[8,8,9,10,10,10],[11,11,9,9,10,10]]

print (grid)
tamanhogrid = len(grid) #
maxregiao = 0 # inicializa o tamanho em 0

gridinit = grid

for i in gridregiao:
    for j in i:
        if j > maxregiao:
            maxregiao = j

tamanhoregiones = [0]*maxregiao

for i in gridregiao:
    for j in i:
        if j != 0:
            tamanhoregiones[j-1] +=1

testeuno()

choice(0,0)
```

Figura 3: Implementação em Python do programa.

3.4. Implementação em Haskell:

Infelizmente não foi possível fazer uma implementação completa do algoritmo em um programa funcional devido a falta de familiaridade do autor com a linguagem Haskell. Porém ainda conseguiu ser implementado todas as estruturas de dados, funções auxiliares e o algoritmo de recursão, apesar de não funcional.

4. DIFICULDADES:


A principal dificuldade apresentada durante o processo de desenvolvimento foi a tradução de um código feito em uma linguagem multiparadigma para uma linguagem funcional como Haskell. O autor sentiu extrema dificuldade na mudança de perspectiva imperativa para funcional.

A falta do comando “for” trouxe desafios em relação à implementação, obrigando o uso de recursão.

Além disso, houve dificuldades na parte da “parada” da função (quando o último elemento da lista é colocado), por isso não houve implementação dessa função. Ademais, houveram diversos problemas com tipagem de código e consertar “bugs”.

E principalmente, há o fato de que o algoritmo para resolução do jogo não é otimizado, e em qualquer tabuleiro mais complexo pode levar horas ou dias para resolver.

5. REFERÊNCIAS:

 AFP 3 - Sudoku II: Initial Solvers - Vídeo explicando um resolvidor de sudoku que foi adaptado para o algoritmo.

<https://stackoverflow.com/questions/5852722/> - Função de Inserção em Lista utilizada no código.

<https://www.educative.io/answers/> - Função Append em lista utilizado no código.