

# Trabalho I

## Implementação de um solucionador para o jogo suguru com programação funcional

Enzo Gomes Sônego – 17202002

Felipe de Campos Santos – 17200441

Thiago Martendal Salvador – 16104594

### O Problema

O trabalho apresenta um solucionador de puzzle suguru, feito utilizando programação funcional com Haskell.

O jogo funciona da seguinte forma:

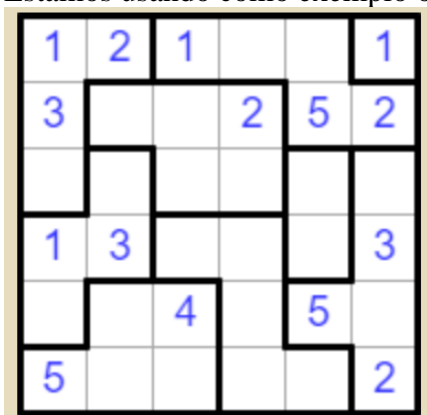
1. É dado um tabuleiro dividido em casas e seccionado em regiões de M casas.
2. O tabuleiro tem tamanho  $N \times N$ , e pode ser representado por uma matriz.
3. Algumas casas do tabuleiro estão preenchidas com um número de 1 a M.
4. O jogo é considerado solucionado quando todas as casas são preenchidas de forma que não existam casas próximas umas das outras com o mesmo número.

### Entrada do Tabuleiro

Para representar o tabuleiro são usadas duas matrizes: suguru e board.

A matriz suguru contém os valores numéricos do tabuleiro. Casas sem números são preenchidas com 0.

Estamos usando como exemplo o Suguru da página [janko](#).



The image shows a 6x6 Suguru puzzle grid. The grid is divided into regions by thick black lines. The numbers in the grid are as follows:

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

Exemplo de entrada com o Suguru acima:

```
let suguru = [
    [1,2,1,0,0,1],
    [3,0,0,2,5,2],
    [0,0,0,0,0,0],
    [1,3,0,0,0,3],
    [0,0,4,0,5,0],
    [5,0,0,0,0,2]]
```

A matriz board contém tuplas que representam as regiões seccionadas do tabuleiro, indicando quais casas fazem parte de cada região. Exemplo:

```
let board = [
    [(1,1), (1,2), (2,1), (3,1)],
    [(1,3), (1,4), (1,5), (2,5), (2,6)],
    [(1,6)],
    [(2,2), (2,3), (2,4), (3,3), (3,4)],
    [(3,2), (4,1), (4,2), (5,1)],
    [(3,5), (4,5)],
    [(3,6), (4,6), (5,5), (5,6), (6,6)],
    [(4,3), (4,4), (5,4), (6,4), (6,5)],
    [(5,2), (5,3), (6,1), (6,2), (6,3)]
]
```

## Solução Abordada

Separamos a resolução do problema em dois grandes blocos, assim facilitando não só a codificação da resolução como também dando uma visualização melhor do que está sendo feito em cada passo.

No primeiro bloco, representado pelo arquivo SuguruSolver, fazemos operações nas matrizes a fim de se atingir o número mínimo de possibilidades apenas com as regras do jogo.

A primeira coisa a se fazer é uma checagem de existência de áreas de tamanho 1, pois por definição essas áreas terão apenas uma possibilidade de preenchimento (o 1), e preenchendo elas já nos livramos de possibilidades 1 ao redor dessas células;

Feito isso, o programa cria uma terceira matriz, no código chamada de “possibilidades”. Essa é uma matriz tridimensional, separada por áreas assim como a matriz board, onde cada sub-lista da matriz principal representa uma área, cada sub-lista da área representa uma célula e esta tem uma lista de possibilidades, a princípio indo de 1 a  $n$ , onde  $n$  é o tamanho da área.

Perceba que num primeiro momento, nem todas as possibilidades representadas nessa matriz são de fato possibilidades. Por exemplo, se tivermos uma área de tamanho 2 com a primeira célula preenchida com 1, a segunda célula estará preenchida com as possibilidades [1,2], apesar de 1 não ser de fato uma possibilidades. Isso é resolvido no próximo passo.

Após a criação dessa matriz de possibilidades populada com listas [1.. $n$ ] (onde  $n$  = tamanho da área), é feita a “limpeza” dessas possibilidades levando em consideração aquilo que já está preenchido em cada área.

Para isso, o algoritmo percorre área por área realizando o seguinte:

- Na área em questão, percorre célula-a-célula checando o tamanho da lista de possibilidades desta;
- Caso ache uma célula com lista de tamanho 1, isso significa que essa célula já está preenchida com sua resposta final. Logo, o número desta célula não pode fazer parte das possibilidades das outras células nessa área. Nesse caso, o algoritmo chama uma outra função que retira da lista de todas as células desta área (menos da célula analisada nessa chamada) o número que foi preenchido na célula.

Nesse ponto do algoritmo, temos uma lista de possibilidades sem repetição de números dentro das áreas. Falta mais um passo de “limpeza”, que é feito logo a seguir

Agora faremos a limpeza ao redor das células do tabuleiro, não importando as áreas (para completar outro passo-regra do jogo, onde não podem haver valores iguais em células adjacentes lado-a-lado, cima-baixo ou diagonal).

Para isso, usaremos as 3 matrizes do programa: Suguru (que contém o tabuleiro original), board (que contém a separação das áreas) e possibilidades (que guarda as listas de possibilidades atualizadas).

Percorrendo área por área, e destas pegando elemento a elemento, checaremos as células apontadas por estes (visto que o elemento mais básico da matriz board é uma dupla que guarda o index da célula da área analisada).

Tendo “em mãos” o index (x,y) da célula, usamos este para checar se esta célula tem uma possibilidade única, assim indicando que esta já está com o valor final, e as células ao redor não poderão ter esse valor. Caso não haja um valor único, fazemos essa mesma chamada para a próxima célula, mas se houver um valor único realizamos os seguintes passos:

- Usaremos o index (x,y) desta célula para ver em qual posição relativa essa célula se encontra:

Esquerda-cima	cima	Direita-cima
Esquerda	Centro	Direita
Esquerda-baixo	baixo	Direita-baixo

Assim saberemos quantas células temos em volta desta (ex: uma célula centro tem células ao redor dela toda, já uma célula esquerda-cima tem células apenas na direita e embaixo) assim evitamos um acesso out of bounds.

Agora, com o index (x,y) e com a posição relativa da célula, usaremos de soma e subtração de 1 unidade para acessar as células ao redor, de acordo com sua posição relativa e retirando das possibilidades destas células o número que está na célula analisada.

Assim, ao chegarmos ao final do tabuleiro, teremos uma lista de possibilidades atualizadas, com as possibilidades mínimas para este tabuleiro.

Aqui, entramos no segundo grande bloco: TrialAndError

A idéia aqui é: agora que temos as possibilidades mínimas, selecionaremos uma primeira possibilidade, e a partir dela vamos refazer as “limpezas” acima, tirando primeiro essa possibilidade escolhida das outras células dessa área e depois tirando das células ao redor. Feito isso, partiremos para escolher a próxima possibilidade e fazer o mesmo.

Ao chegar no final do tabuleiro ou encontrar uma célula vazia, teremos uma matriz de possibilidades onde em todas as células alguma possibilidade foi escolhida ou (no caso da célula vazia) alguma possibilidade foi escolhida erroneamente. Caso seja o primeiro, checamos se a matriz é uma solução, se for retornamos ela, se não for retornamos ao ponto de partida da escolha e refaremos tudo, agora começando pela escolha de outro elemento.

Fazemos isso até que se ache o resultado ou se esgotem as escolhas, nesse caso indicando que não existe solução para a matriz adicionada.

## Resultados:

Segundo o site de onde a matriz foi retirada, o resultado correto segue:

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

E o resultado obtido pelo programa foi:

```
[
[[1],[2],[3],[4]],
[[1],[3,4],[3,4],[5],[2]],
[[1]],
[[5],[3,4],[2],[1,4],[3,4]]
,[[2],[1],[3],[4]],
[[1],[2]],
[[4],[3],[5],[1],[2]],
[[5],[3],[1],[2],[4]],
[[2],[4],[5],[1,3],
[1,3]]
```

1	2	1	3 ou 4	3 ou 4	1
3	5	3 ou 4	2	5	2
4	2	1 ou 4	3 ou 4	1	4
1	3	5	3	2	3
4	2	4	1	5	1
5	1 ou 3	1 ou 3	2	4	2

Ou seja, de 21 células para se achar o resultado, nosso programa achou o resultado para 2/3 na primeira passada.

## Dificuldades e Organização

As dificuldades mais presentes no trabalho foram pelo uso de uma linguagem que nenhum dos integrantes do grupo tinha experiência, e percebemos que é uma linguagem que não é muito utilizada, não contando com tanta variedade de materiais auxiliares pela internet. Tivemos que nos apoiar quase que totalmente na documentação dela (que um dos nossos integrantes (Felipe) acha pouco didática comparada à documentação de outras linguagens).

Encontramos bastante dificuldade também de usar libs ou módulos já existentes para a linguagem, acabando quase que nos deixando refém da criação do zero de tudo.

A organização do desenvolvimento do grupo foi feita por whatsapp, onde mantínhamos o grupo atualizado com as alterações que havíamos feito, e com o discord como apoio para momentos de encontro para se ajudar na resolução, e usando do git para o versionamento do programa. Devido à situação atípica que estamos passando nesse ano, e a ainda adaptação nessa modalidade de EAD, as coisas ficavam um pouco bagunçadas as vezes, mas o tempo nos permitiu uma organização um pouco melhor a cada dia, e com a experiência aperfeiçoamos.