Universidade Federal de Santa Catarina
Paradigmas de Programação - INE5416

Resolvedor de Suguru - LISP

Alunos: Arthur Lisboa - 18100518
Igor May Wensing - 17203362

Florianópolis, abril de 2021

1 - Análise do Problema Descrito

Escolhemos o jogo Suguru para desenvolver um resolvedor em Haskell. Este jogo possui um tabuleiro quadrado de largura variável, com áreas formadas por campos (quadradinhos), e alguns valores dispostos inicialmente no tabuleiro. As regras do jogos são:

- Você deve inserir um valor em cada campo do tabuleiro
- Uma área de N campos deve conter todos os valores de 1 a N exatamente uma vez
- Os números em campos adjacentes, ortogonalmente ou diagonalmente, devem ser diferentes

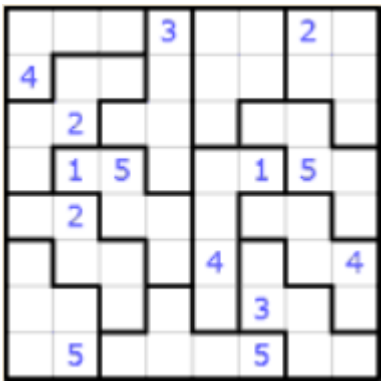

Fig 1. Exemplo de um tabuleiro inicial 8x8

2 - Solução

Para resolver o problema, o primeiro passo foi escolher uma estrutura de dados para representar o tabuleiro, para isso utilizamos duas matrizes, uma para representar a área e outra para representar os valores de cada posição (x,y) do tabuleiro.

```
MATRIZ DE VALORES:                    MATRIZ DE ÁREAS:

0  0  0  3  0  0  2  0                0  0  0  1  2  2  3  3
4  0  0  0  0  0  0  0                0  4  4  1  2  2  3  3
0  2  0  0  0  0  0  0                4  4  1  1  2  7  7  3
0  1  5  0  0  1  5  0                4  5  5  1  6  6  7  7
0  2  0  0  0  0  0  0                8  8  5  5  6  11 11 7
0  0  0  0  4  0  0  4                9  8  8  5  6  12 11 11
0  0  0  0  0  3  0  0                9  9  8  10 6  12 12 11
0  5  0  0  0  5  0  0                9  9  10 10 10 10 12 12
```

Fig 2. Tabuleiro da **Fig 1** representado em duas matrizes

O próximo passo foi desenvolver o método *resolvedor*, para isso utilizamos a técnica de backtracking, como mostra a figura abaixo:

```
56  ;;Função que de fato resolve o tabuleiro.
57  (defun resolvedor (valores areas)
58      (if
59          (block resolvedor-block
60              (dotimes (y largura)
61                  (dotimes (x largura)
62                      (if (= (aref valores y x) 0)
63                          (progn
64                              (dotimes (valor (numElementosArea areas y x))
65                                  (if (verificador valores areas y x (+ valor 1))
66                                      (progn
67                                          (setf (aref valores y x) (+ valor 1))
68                                          (resolvedor valores areas)
69                                          (setf (aref valores y x) 0)
70                                      )
71                                  )
72                              )
73                              (return-from resolvedor-block NIL)
74                          )
75                      )
76                  )
77              )
78              T
79          )
80          (imprimirTabuleiro valores)
81      )
82  )
```

Fig 3. Trecho de código relacionado ao resolvedor.

A função *resolvedor* tenta achar soluções válidas para o problema, caso não haja nenhuma solução não imprime.

O método começa percorrendo uma linha y do tabuleiro e uma coluna x do tabuleiro, em seguida verifica se uma posição (y,x) é vazia, ou seja, possui valor 0.

Logo depois, a partir dos valores possíveis que vão de 1 até o número de elementos de uma área, calculado pelo método *numElmentosArea,* verifica-se com o método *verificador* se esse valor pode ser escrito na posição (y,x). Caso o valor possa ser escrito, ele escreve e chama a função *resolvedor* para resolver essa nova configuração do tabuleiro e depois retira o valor da posição (y,x) para tentar escrever o próximo valor, tentando assim encontrar todas as soluções existentes.

Se ele encontrou uma posição vazia, significa que não foi encontrada uma solução válida ainda, então retorna NIL e não imprime o tabuleiro.

Se não encontrou posição vazia, então significa que chegamos numa solução que é válida, então retorna T e imprime o tabuleiro.

3 - Entrada e Saída

Nosso programa não possui entrada via terminal, então deve-se digitar o jogo diretamente no código-fonte a largura do tabuleiro, os valores e as áreas de cada posição.

4 - Vantagens e desvantagens

Não vimos nenhuma desvantagem em utilizar a linguagem LISP durante o nosso desenvolvimento. Já a grande vantagem observada ao utilizar LISP é o fato de existir laço de repetição e portanto não dependendo exclusivamente de recursividade.

5 - Mudanças

As principais mudanças foram em como representamos o tabuleiro pois neste trabalho utilizamos duas matrizes de inteiros ao invés de uma lista de tuplas e também do no jeito de implementar o método que resolve o tabuleiro.

6 - Divisão do Trabalho

Achamos que a melhor alternativa para o desenvolvimento do trabalho era nos reunirmos via Discord, onde um integrante tinha o código aberto e compartilhava sua tela. Como as resoluções dependiam muito de discussão e não tanto de trabalho mecânico, o nosso método de resolução se mostrou muito produtivo.

7 - Dificuldades e Solução

Como aconteceu no trabalho anterior, a principal dificuldade não estava na criação do algoritmo em si, mas sim em saber como a linguagem funciona, então vários problemas que em uma outra linguagem resolveríamos rápido, demoramos mais tempo para resolver. Um exemplo disso foi quando algumas vezes nós nos perdemos onde colocar os parênteses.

Já para a solução, tivemos que pesquisar um pouco em como implementar backtracking em LISP. Para isso usamos como referência um resolvedor de sudoku que utilizava backtracking (https://stackoverflow.com/questions/6842070/lisp-backtracking) mas adaptando para o nosso jogo e também utilizamos o site https://www.tutorialspoint.com/lisp/index.htm para saber mais sobre a linguagem.