UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
CIÊNCIA DA COMPUTAÇÃO

Artur Luiz Rizzato Toru Soda

Relatório - Trabalho 2 - Programação Funcional - Lisp
Kojum

Florianópolis
2024

1. INTRODUÇÃO

Este trabalho tem como objetivo aplicar os conhecimentos aprendidos na disciplina Paradigmas de Programação, criando um resolvedor para uns dos mesmos três puzzles mostrado no enunciado do trabalho I, entretanto, dessa vez deveria ser escolhido uma das linguagens mostrado no enunciado deste trabalho II.

Dessa forma, o problema escolhido neste trabalho foi o mesmo do Trabalho I, Kojum, cujo o funcionamento é da seguinte forma: Dado um tabuleiro dividido em diversas regiões, um número deve-se ser inserido em cada campo, de modo que: Cada região de tamanho N contenha números de 1 à N exatamente uma vez; Campo adjacentes ortogonalmente devem ter valores diferentes; E por fim, se dois campo são verticalmente adjacentes na mesma região, o valor de cima deve ser maior que o de baixo.O puzzle é resolvido quando o tabuleiro é completo, respeitando as três regras supracitadas.

Além disso, a linguagem escolhida para a implementação foi Common Lisp.

2. DESENVOLVIMENTO

2.1. BACKTRACKING

Tendo noção das regras do puzzle, a estratégia utilizada para a resolução foi o backtracking, um algoritmo para resolução de problemas que envolve encontrar uma solução tentando diferentes opções e desfazendo-as se levarem a uma conclusão incorreta, comumente usado em situações em que é preciso explorar múltiplas possibilidades. Assim o algoritmo volta ao ponto de decisão anterior e explora um caminho diferente até que uma solução seja encontrada ou todas as possibilidades tenham sido esgotadas.

2.2. HASKELL

Como o puzzle escolhido, neste trabalho, foi o mesmo do trabalho passado, a lógica de solução usada é a mesma da realizada na implementação em Haskell.

Dessa forma, para a resolução bastou a tradução do código fonte de Haskell para LISP. Assim, como as duas linguagens seguem o paradigma funcional, não foi encontrada muita dificuldade na transcrição.

2.3. LISP - SOLUÇÃO

No começo de sua execução, o programa lê do teclado qual tabuleiro deve ser resolvido, carregando então a matriz de valores e de regiões.

```
; Le arquivos e retona matrizes de valores e regioes
(defun readBoard (path)
    (setq contentFile (open (concatenate 'string (concatenate 'string "Entradas/" path) "/numbers.txt")))
    (setq regionsFile (open (concatenate 'string (concatenate 'string "Entradas/" path) "/regions.txt")))

    (setq valuesMatriz (map 'list (lambda (arr) (map 'list #'symbolToInt arr)) (read contentFile)))
    (setq regionsMatriz (read regionsFile))

    (list valuesMatriz regionsMatriz)
)


; Transforma '*' em -1
(defun symbolToInt (a)
    (if (typep a 'symbol) -1 a)
)

(defun main()
    ; Le do usuario qual tabuleiro resolver
    (setq path (string (read)))
```

Em seguida o tabuleiro é montado, onde cada campo é representado por uma *Posição*: Uma estrutura que guarda informações como, seu valor, bloqueio de cima, baixo, esquerda e direita, e a região em que pertence.

```
(defstruct Pos
    value
    upBorder
    downBorder
    leftBorder
    rightBorder
    region
)


; Instancia posicoes, definindo seu valor, bloqueios e regiao
(defun makePosition (i j v rm len)
    (let ((reg (nth j (nth i rm))))
        (make-Pos
            :value v
            :upBorder (if (= i 0) T (string/= reg (nth j (nth (- i 1) rm))))
            :downBorder (if (= i (- len 1)) T (string/= reg (nth j (nth (+ i 1) rm))))
            :leftBorder (if (= j 0) T (string/= reg (nth (- j 1) (nth i rm))))
            :rightBorder (if (= j (- len 1)) T (string/= reg (nth (+ j 1) (nth i rm))))
            :region reg
        )
    )
)
```

Em seguida, com o tabuleiro inicial montado, a função *solve* é chamada, que recebendo o tabuleiro como parâmetro, é retornado um *booleano*, ou seja, caso não for possível resolver, a função retorna *Nill*, mas caso contrário, *T* é retornado. Assim, tendo conhecimento do backtracking, sua lógica é da seguinte forma: a função percorre por todos os campos do tabuleiro, e nas casas em que ainda não possuem um valor, é feita a verificação se um número candidato é válido na célula. Caso não seja, tentamos atribuir outro valor ao campo, e se caso as opções de candidatos acabarem, voltamos uma casa para trás. Mas, se for válido, atribuímos esse valor ao campo e tentamos resolver a partir deste novo tabuleiro, se for resolvível retornamos *T*, senão, retiramos o valor atribuído, e tentamos o próximo candidato.

Dessa forma, esses passos são realizados, até que o tabuleiro seja resolvido, ou até que todas as possibilidades acabem.

```
12  ; Logica para resolucao do tabuleiro
13  (defun solveAux (i j guess board len regionHT)
14      (cond
15          ; Caso passe da ultima linha -> problema resolvido -> retorna verdadeiro
16          ((= i len) T)
17          ; Caso passe da ultima coluna -> passa para a proxima linha
18          ((= j len) (solveAux (+ i 1) 0 guess board len regionHT))
19          ; Caso posicao ja possui um valor -> passe para o proximo da coluna
20          ((/= (Pos-value (getPosition i j board)) -1) (solveAux i (+ j 1) guess board len regionHT))
21          ; Senao tenta atribuir um valor a posicao:
22          (t
23              ; Verifica chute
24              ; Se chute chegar a 10, cancela e temos que voltar uma passo
25              (if (= guess 10) nil
26                  ; Se ainda nao chegou ao ultimo chute
27                  (let ((pos (getPosition i j board))
28                        (regionList (gethash (Pos-region (getPosition i j board)) regionHT)))
29
30                      ; Verifica se o chute eh valido nessa posicao
31                      (cond
32                          ; Se nao for valido passamos para o prossimo candidato
33                          ((not (validate i j guess regionList board)) (solveAux i j (+ guess 1) board len regionHT))
34                          (t
35                              ; Se for valido, atribuimos o candidato a posicao, e atulizamos o hash table das posicoes
36                              (setf (Pos-value pos) guess)
37                              (setq regionList (cons guess regionList))
38                              (setq regionList (remove -1 regionList :count 1))
39                              (setf (gethash (Pos-region pos) regionHT) regionList)
40
41                              ; Entao tentamos resolver a partir do novo tabuleiro
42                              (cond
43                                  ; Se for resolvivel retornamos verdadeiro
44                                  ((solve board) T)
45                                  (t
46                                      ; Se nao for, retiramos candidato atribuido, e tentamos atribuir proximo chute
47                                      (setf (Pos-value pos) -1)
48                                      (setq regionList (remove guess regionList :count 1))
49                                      (setq regionList (cons -1 regionList))
50                                      (setf (gethash (Pos-region pos) regionHT) regionList)
51                                      (solveAux i j (+ guess 1) board len regionHT)
52                                  )
```

Para verificar se a casa possui um valor válido, respeitando as regras já citadas neste relatório, a função *validate* é chamada: recebendo como parâmetros, as coordenadas (I, J) da casa a ser verificada, o valor candidato a ser atribuído, e a lista de valores existentes na região, retorna um booleano indicando a validação.

```
63      ; Verifica se chute eh valido para a posicao
64      (defun validate (i j guess regionList board)
65          (let ((pos (getPosition i j board))
66                (len (getLen board))
67                (regionLen (getlen regionList)))
68              ; Se valor ja esta na regiao
69              (and (notInArr guess regionList)
70              ; Se o valor eh maior do que a regiao permite
71              (and (<= guess regionLen)
72              ; Verifica se adequa a posicao de cima
73              (and (verifyAbovePosition i j guess pos board)
74              ; Verifica se adequa a posicao de baixo
75              (and (verifyBellowPosition i j guess pos board len)
76              ; Verifica se adequa a posicao da esquerda
77              (and (verifyLeftPosition i j guess board)
78              ; Verifica se adequa a posicao da direita
79                   (verifyRightPosition i j guess board len))))))
80          )
81      )
```

Por fim, caso o tabuleiro passado seja possível de resolver, é feita a impressão do puzzle resolvido.

```
24      (defun main()
25          ; Le do usuario qual tabuleiro resolver
26          (setq path (string (read)))
27
28          ; Monta matrizes de valores e regioes
29          (setq matrizes (readBoard path))
30          (setq valuesMatriz (nth 0 matrizes))
31          (setq regionsMatriz (nth 1 matrizes))
32
33          ; Monta Tabuleiro de posicoes
34          (setq board (makeBoard valuesMatriz regionsMatriz 0))
35
36          (write-line "")
37
38          ; Imprime tabuleiro antes de ser resolvido
39          (printBoard board)
40
41          (write-line "")
42
43          ; Caso for possivel resolver o tabuleiro, ele eh imprimido
44          ; Caso contrario informa que tabuleiro nao possui solucao
45          (cond
46              ((not (solve board)) (write-line "Tabuleiro nao possui solucao"))
47              (t
48                  (printBoard board)
49                  (write-line "")
50                  (prettyPrint board)
51              )
52          )
53      )
```

3. EXECUÇÃO

Como já dito anteriormente, o problema a ser resolvido é lido do teclado, que é o nome do tabuleiro que leva ao diretório onde possui dois arquivos textos, um com os valores de cada campo, e outro indicando as regiões (indicadas por caracteres).

```
1     ((5 * 2 * 2 * 3 1 3 1)
2     (* 4 * 1 * 5 * 5 * 4)
3     (7 5 1 7 * * 3 1 3 *)
4     (* 4 * * * * * * * 3)
5     (2 * 3 4 * 2 * * 4 *)
6     (5 * 2 * 6 * * * * *)
7     (* 1 3 * 1 * * 4 * 3)
8     (6 7 * 3 * 1 4 * * 1)
9     (4 * 3 * 4 * * * * 3)
10    (* 1 * 2 * 6 2 * 2 1))
```

```
1     ((a b b b c c c c d d)
2     (a a a b e e f f d f)
3     (i i a e e g h f f f)
4     (i i e e j g h h h l)
5     (i i i e j j k l l l)
6     (m m n n n j o o p p)
7     (m m m n n q r s p p)
8     (v v m n q q r s t t)
9     (v v w w w w r s s u)
10    (v v v w w w r r u u))
```

Logo, com o tabuleiro resolvido é feito a impressão do estado inicial do quadro, depois uma impressão simplificada de cada campo, e por fim, uma impressão mais detalhada da solução, com suas regiões separadas.

```
artur@desktop-artur ~/UFSC/Paradigmas_de_Programacao/Trabalho_2 <main>
➤ ./result
T10x10_1

5 . 2 . 2 . 3 1 3 1
. 4 . 1 . 5 . 5 . 4
7 5 1 7 . . 3 1 3 .
. 4 . . . . . . . 3
2 . 3 4 . 2 . . 4 .
5 . 2 . 6 . . . . .
. 1 3 . 1 . . 4 . 3
6 7 . 3 . 1 4 . . 1
4 . 3 . 4 . . . . 3
. 1 . 2 . 6 2 . 2 1

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

|--------------------|
|5|3 2 4|2 4 3 1|3 1|
| |---| |-------| |-|
|2 4 3|1|3 5|6 5|2|4|
|---| |-| |---| |-| |
|7 5|1|7 1|2|3|1 3 2|
| | |-| |-| | |-----|
|6 4|2 6|4|1|2 4 1|3|
| | |-| | |-------| |
|2 1 3|4|3 2|1|2 4 1|
|---------| |-------|
|5 6|2 5 6|1|2 1|2 4|
| | |-| | |-----| | |
|4 1 3|4 1|3|5|4|1 3|
|---| | |-| | | |---|
|6 7|2|3|2 1|4|3|2 1|
| | |-------| | |---|
|4 2|3 5 4 7|3|2 1|3|
| | |-| | | | |---| |
|3 1 5|2 1 6|2 1|2 1|
|--------------------|
```

4. CONCLUSÃO E DIFICULDADES

Considerando os tópicos supracitados, é possível concluir que o projeto proposto, foi uma ótima prática, para o um melhor conhecimento da linguagem funcional LISP. Entretanto, mesmo que a lógica de solução seja a mesma do trabalho I, e também, Haskell e LISP seguem o mesmo paradigmas, as linguagens possuem suas diferenças.

Dessa forma, tive dificuldade na tradução das linguagens, pois algumas estruturas e funções existentes em uma, não existem na outra, e vice e versa. Então, foi preciso pesquisas por diferentes alternativas, e assim analisarmos qual o melhor para o devido desafio.