Trabalho II – Resolvedor de Suguru em LISP

Arthur Machado Capaverde – 17207215

Implementação

O código é dividido em 6 partes: a configuração da entrada (feita diretamente no código fonte), 3 testes, a busca da solução utilizando backtracking com os testes previamente implementados e a parte final da impressão do resultado na tela.

Como configurar a entrada

```
;O tabuleiro que queremos resolver, zeros representam espaços vazios
(setf tabuleiro (list    0  3  0  2  0  3
                         4  0  4  0  1  0
                         0  0  0  5  0  4
                         3  5  0  0  0  0
                         0  0  0  2  0  0
                         5  2  0  0  4  0))

; O tabuleiro com as areas definidas atraves de tuplas
(setf tabuleiroDiv (list    '(1 0) '(1 3) '(1 0) '(2 2) '(2 0) '(2 3)
                            '(3 4) '(4 0) '(1 4) '(5 0) '(2 1) '(2 0)
                            '(3 0) '(4 0) '(4 0) '(5 5) '(5 0) '(5 4)
                            '(3 3) '(3 5) '(4 0) '(6 0) '(6 0) '(5 0)
                            '(7 0) '(3 0) '(4 0) '(6 2) '(6 0) '(8 0)
                            '(7 5) '(7 2) '(7 0) '(7 0) '(6 4) '(8 0)))
```

```
;n é a dimensao do tabuleiro(tabuleiro tera tamanho nxn)
(setq n 6)
```

Os elementos do tabuleiro devem ser colocados na matriz *tabuleiro* e os elementos do tabuleiro junto com identificadores da área a que pertencem devem ser inseridos em forma de tupla (identificador, elemento) na matriz *tabuleiroDiv*. A dimensão do tabuleiro também deve ser inserida em *n*.

Primeiro teste: lados e diagonais

```
;--testa se o quadrado ao sudeste tem um número igual
(defun seOk(i lista)
    (if (or (>= i (* n (- n 1))) (= (mod (+ i 1) n) 0)) T
        (if (= (nth (+ (+ i 1) n) lista) 0) T
            (if (= (nth i lista) (nth (+ (+ i 1) n) lista))
                NIL
                T)))
)

;--testa se o quadrado ao sudoeste tem um número igual
(defun swOk(i lista)
    (if (or (>= i (* n (- n 1))) (= (mod i n) 0)) T
        (if (= (nth (- (+ i n) 1) lista) 0) T
            (if (= (nth i lista) (nth (- (+ i n) 1) lista))
                NIL
                T)))
)

;--testa se algum dos quadrados em volta tem um número igual
(defun sidesOk(i lista)
    (if (and (rightOk i lista) (leftOk i lista) (upOk i lista) (downOk
)
```

Este teste checa a regra de que o número inserido não pode ser igual a um número nos seus lados ou diagonais, cada função checa para uma direção específica e sidesOk utiliza todas elas para checar todas as direções.

Segundo teste: números na mesma área

```
;--testa se duas tuplas são iguais e retorna 1 se sim e 0 se não
(defun isEqual(x y)
    (if (and (= (nth 0 x) (nth 0 y)) (= (nth 1 x) (nth 1 y)))
        1
        0)
)

;--soma os elementos de uma lista
(defun somaLista (lista)
    (if lista
      (+ (car lista) (somaLista (cdr lista)))
      0)
)

;--testa se tem dois números iguais na área do indice i
(defun areaOk(i lista)
    (if (= (somaLista (mapcar (lambda (x) (isEqual x (nth i lista))) lista)) 2)
        NIL
        T)
)
```

Este teste checa a regra de que o número inserido não pode ser igual a um número na sua área/seção, isEqual retorna 1 caso os elementos testados sejam iguais e 0 caso não seja. somaLista soma os elementos de uma lista e areaOk vai somar todos os elementos da matriz retornada por mapcar e caso a soma seja igual a 2 irá retornar False. O número 2 foi utilizado pois significa que mapcar encontrou o próprio elemento sendo testado (+1) e um elemento igual a ele (+1=2).

Terceiro teste: tamanho da área

```
;--testa se duas tuplas tem o mesmo primeiro elemento
(defun isEqualp (x y)
    (if (= (nth 0 x) (nth 0 y))
        1
        0)
)

;--retorna quantos quadrados tem a área do indice i
(defun areaSize(i lista)
    (somaLista (mapcar (lambda (x) (isEqualp x (nth i lista))) lista))
)

;--testa se o elemento e é maior do que a quantidade de quadrados da área do indice i
(defun possibilitiesOk(i lista)
    (if (< (nth 1 (nth i lista)) (+ (areaSize i lista) 1))
        T
        NIL)
)
```

Este teste checa se o número inserido está no intervalo (1:N) onde N é o tamanho da área/seção do índice onde o número foi inserido. A função mapcar é utilizada com isEqualp para retornar uma matriz com 1s nos indíces que pertencem a área/seção e 0s nos outros. AreaSize então soma todos os elementos desta matriz para encontrar N e possibilitiesOk checa se o número inserido é menor do que N, não é necessário testar se o número é maior do que 0 pois 0 não está na matriz *possibilities* declarada no começo do código que contém todos os elementos inseríveis [1, 2, 3, 4, 5, 6, 7, 8, 9].

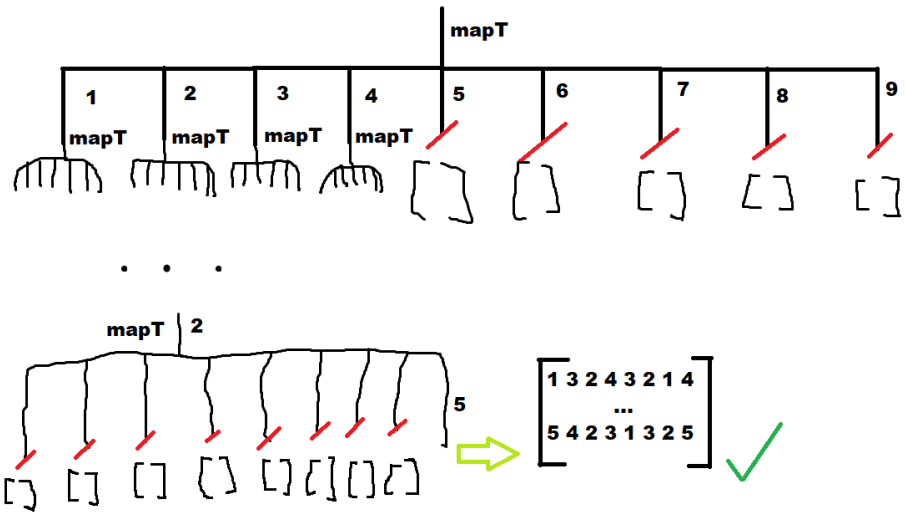
Encontrando a solução

```
;--aplica todos os testes previamente implementados
(defun tester(i lista listaDiv)
    (and (sidesOk i lista) (areaOk i listaDiv) (possibilitiesOk i listaDiv))
)

;--testa se o tabuleiro foi completamente preenchido
(defun isEnd(i lista)
    (if (and (= i g) (/= (nth i lista) 0))
        T
        NIL)
)

;--encontra o próximo espaço vazio na matriz
(defun nextBlank(i lista)
    (cond   ((= i g) g)
            ((= (nth (+ i 1) lista) 0) (+ i 1))
            (T (nextBlank (+ i 1) lista)))
)
```

```
;--retorna um novo tabuleiro com o elemento x inserido no indice i
(defun getTesterMap(x i lista)
    (concatenate 'list (concatenate 'list (subseq lista 0 i) (list x)) (subseq lista (+
)

;--retorna um novo tabuleiroDiv com o elemento x inserido no indice i
(defun getTesterDivMap(x i listaDiv)

    (concatenate 'list (concatenate 'list (subseq listaDiv 0 i) (list (list (nth 0 (nth
)

;--aplica a função solver com todos os elementos da matriz x(possibilities)
(defun mapT(i lista listaDiv)
    (mapcar (lambda (x) (solver x i lista listaDiv)) possibilities)
)

;--funcao recursiva que encontra a solução do suguru
(defun solver(x i lista listaDiv)
    (if (tester i (getTesterMap x i lista) (getTesterDivMap x i listaDiv))
        (if (isEnd i (getTesterMap x i lista))
            (getTesterMap x i lista)
            (mapT (nextBlank i lista) (getTesterMap x i lista) (getTesterDivMap x i lis
            ;(mapcar (lambda (x) (solver x (nextBlank i lista) (getTesterMap x i lista)
        NIL
    )
)
```

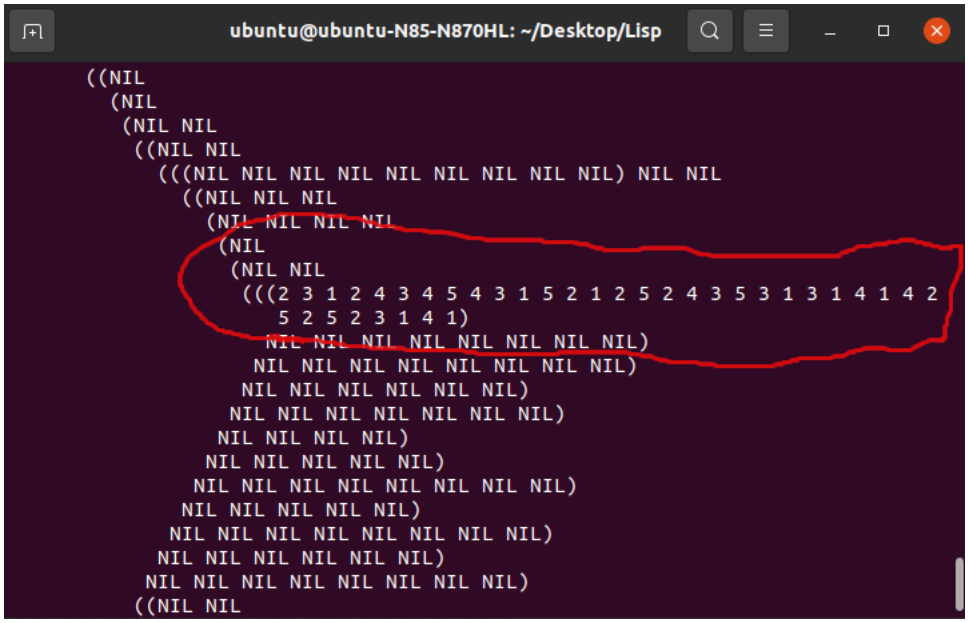
O algoritmo utilizado utiliza *backtracking* para encontra a solução, primeiro ele procura o primeiro espaço vazio da matriz (=0) e então insere sucessivamente os dígitos de 1 a 9 começando pelo 1. ele então testa se o elemento inserido viola alguma das regras do suguru (lados, área, etc.), caso viole ele retorna NIL.

Caso o elemento inserido passe no teste o algoritmo checa se o tabuleiro está completamente preenchido, se sim, então encontramos a solução e retornamos a matriz construída até agora. Se não, ele procura o próximo espaço vazio e recomeça a operação de inserir elementos. Segue um desenho para tentar ilustrar este processo:



Devido ao fato de uma matriz vazia ser equivalente a NIL em LISP, o resultado é uma matriz de matrizes com variados preenchimentos de NIL, porém a solução está printada como um elemento destas matrizes e é facilmente encontrado como mostrado na seção seguinte.

Interpretando a solução



Infelizmente devido aos NILs eu não consegui inserir as quebras de linha para formatar a solução do suguru, porém o resultado está correto, baste lê-lo na ordem em que os números aparecem no suguru: de cima para baixo, da esquerda para direita.

Conclusão

Este trabalho não foi tão difícil quanto o primeiro pois usei meu código em haskell como base, porém senti falta de algumas funções úteis para trabalhar com listas (sum, take, drop, etc.) e não há tantos recursos na internet para LISP como para outros linguagens, passei muito tempo pesquisando para achar algumas funções que eu precisava (concatenate, subseq). Eu peço desculpas pela inconveniência de ter que encontrar a solução no meio dos NILs mas por serem diversas matrizes em formatos diferentes ficou dĩficil de usar algo como (remove NIL matriz) para retirar os NILs, já que, pelo meu entendimento, ((NIL) NIL NIL) é diferente de NIL.