Exercícios da cadeira de Inteligência Artificial

Helena Sofia Pinto
João Cachopo
Daniel Gonçalves
Carlos Lopes António
Inês Lynce
Pedro Matos

Ano Lectivo 2001/2002

Prefácio

Este documento reúne os exercícios propostos nas aulas práticas da cadeira de Inteligência Artificial da Licenciatura de Engenharia Informática e de Computadores do Instituto Superior Técnico. Esta compilação destina-se a ser usada na referida cadeira no ano lectivo 2001/2002.
Alguns destes exercícios correspondem à tradução de exercícios propostos no livro de texto base da cadeira:

- Stuart Russel e Peter Norvig, *Artificial Intelligence: A Modern Approach*, Englewood-Cliff, N.J.: Prentice-Hall, 1995.

Sempre que tal se verifique aparece associado ao número do exercício uma indicação do número que o exercício tem no referido livro.
O exercício 9.1 foi retirado do livro:

- C-L. Chang e R.C-T. Lee, *Symbolic Logic and Mechanical Theorem Proving*, New-York, N.Y.: Academic Press, 1973.

H. Sofia Pinto
(editora)

1 Lisp

Exercício 1.1
Escreva as seguintes expressões em notação Lisp, considerando a prioridade normal das operações:

1. 5 + 76 - 8
2. 17 - 45 * 23
3. (17 - 45) * 23
4. 23 - 57 * 34 / 687
5. 15 * 7 + 48 / 5 * 10 + 8 / 4

Exercício 1.2
Quais os resultados obtidos ao avaliar as seguintes expressões no seu ambiente Lisp:

1. `(and (= 3 4) (< 3 4) (> 3 4))`
2. `(or (= 3 4) (< 3 4) (> 3 4))`
3. `(and (+ 3 4) (- 3 4) (* 3 4))`
4. `(or (+ 3 4) (- 3 4) (* 3 4))`

Exercício 1.3
Defina a função `max` que recebe dois números e devolve o maior deles.

```
>(max 5 9)
9
```

Exercício 1.4
Defina a função `max3` que recebe três números e devolve o maior deles:

1. Utilizando `if`'s.
2. Utilizando `cond`.
3. Utilizando a função `max` definida no exercício anterior.

Exercício 1.5
Defina a função `soma` que recebe dois números inteiros não negativos e retorna a sua soma. Apenas pode usar recursão, a forma especial `if` e as funções `1+`, `1-` e `zerop`.

```
> (soma 3 4)
7
```

Exercício 1.6
Defina a função `potencia` que recebe dois números e calcula o valor de elevar o primeiro (a base) ao segundo (o expoente). Assuma que o expoente é sempre um inteiro >= 0.

```
> (potencia 3 2)
9
>(potencia 5/10 3)
1/8
>(potencia (/ 1 3) 3)
1/27
```

Exercício 1.7
Qual o resultado das expressões seguintes:

```
>(cons 2 ())

>(cons nil nil)

>(cons '(1 2 3) '(a b c))

>(cons 1 '(1 2 3))

>(first '(1 2 3))

>(rest '(1 2 3))

>'(+ 1 2 3)

>(list 1 2 3)

>(list 'a 'b 'c)

>(list 1 (+ 1 2 3) 3)

>(list 1 '(+ 1 2 3) 3)

>(append '(1 2 3) '(a b c))
```

Exercício 1.8
Defina a função `soma-1` que recebe uma lista de números e retorna uma lista de números cujos elementos são os elementos da lista argumento adicionados de uma unidade.

```
>(soma-1 '(1 2 3))
(2 3 4)
```

Exercício 1.9
Defina a função `soma-n` que recebe um número, n, e uma lista de números e retorna uma lista de números cujos elementos são os elementos da lista argumento adicionados de n.

```
>(soma-n 3 '(1 2 3))
(4 5 6)
```

Exercício 1.10
Por exemplo, para produzir a lista (1 2 3) devemos introduzir no avaliador:

```
>'(1 2 3)
```

Escreva expressões no avaliador de forma a produzir os seguintes resultados:

1. '(1 2 3)
2. (1 '2 3)
3. ((1 2 3) (3 5) (8))

Exercício 1.11
Tente que o avaliador retorne a lista ((1 2 3) (3 5) (8)) usando para tal apenas a função `cons`, `nil` e os números.

Exercício 1.12
Defina a função `comprimento` que recebe uma lista de elementos e retorna o número de elementos dessa lista.

```
>(comprimento '(1 2 3))
3
```

Exercício 1.13
Defina a função `junta` que recebe duas lista de elementos e retorna a lista resultante de juntar a primeira à segunda lista.

```
>(junta '(a b c) '(1 2 3))
(a b c 1 2 3)
```

Exercício 1.14
Defina a função `inverte` que recebe uma lista de elementos e retorna uma lista com os elementos da lista argumento pela ordem inversa.

```
>(inverte '(a b c))
(c b a)
```

Exercício 1.15
Defina a função `membro` que recebe um elemento e uma lista de elementos e retorna verdade se o elemento pertence à lista e falso em caso contrário.

```
>(membro 'd '(a b c))
nil
>(membro 'c '(a b c))
t
```

Exercício 1.16
Defina a função `retira` que recebe um elemento e uma lista de elementos e retorna uma lista que corresponde à lista argumento à qual foram retiradas todas as ocorrências do elemento.

```
>(retira 'c '(c a b c))
(a b)
```

Exercício 1.17
Defina a função `conta-ocorrencias` que recebe um elemento e uma lista de elementos e retorna o número de vezes que esse elemento ocorre na lista argumento.

```
>(conta-ocorrencias 'd '(a b c))
0
>(conta-ocorrencias 'c '(a b c))
1
>(conta-ocorrencias 'c '(c a b c))
2
```

Exercício 1.18
Defina a função `igual-comprimento` que recebe duas lista de elementos e retorna verdade se ambas as listas tiverem igual comprimento e falso caso contrário. Não pode usar a função length ou uma função auxiliar equivalente a length.

```
> (igual-comprimento '(0) '(1))
T
>(igual-comprimento '(1 2) '(1 2 3))
NIL
```

Exercício 1.19
Defina a função `posicao` que recebe um elemento e uma lista de elementos e devolve a posição em que o elemento ocorre na lista (sendo a primeira posição a zero). Se o elemento não existir na lista devolve NIL.

```
>(posicao 5 '(1 2 3))
NIL
>posicao 3 '(1 2 3))
2
>(posicao 2 '(1 2 3 2 1 2))
1
>(posicao 'defun '(defun xpto (ola) (+ ola 3)))
0
```

Exercício 1.20
Defina a função `substitui` que recebe dois elementos e uma lista de elementos e devolve a lista com todas as ocorrências do primeiro elemento substituídas pelo segundo.

```
>(substitui 5 7 '(1 2 3))
(1 2 3)
>(substitui 3 5 '(1 2 3))
(1 2 5)
>(substitui 2 7 '(1 2 3 (2) 1 2))
(1 7 3 (2) 1 7)
>(substitui 'xpto 4 '(defun xpto (ola) (+ ola 3)))
(defun 4 (ola) (+ ola 3))
```

Exercício 1.21
Defina a função `interseccao` que recebe duas listas sem elementos repetidos e devolve uma lista que contém todos os elementos em comum nas duas listas. Sugestão: Utilize a função `member` do Common Lisp.

```
>(interseccao '(14 52 37 28) '(28 76 12 52))
(28 52)
>(interseccao '(xpto zky bdo) '(pois XPTO ola))
(xpto)
>(interseccao '(xpto zky bdo) '(fred zyc ola))
NIL
```

Exercício 1.22
Qual o valor da seguinte expressão:

```
>(let ((x 10))
    (+ (let ((x 20))
         x)
       x))
```

Exercício 1.23
Defina a função `profundidade` que recebe uma lista e devolve um número que indica qual é o nivel mais profundo de listas dentro dessa lista, aumentando de um sempre que se entra para dentro de uma lista. Se a lista não contém listas devolve 0.

```
>(profundidade '(1 2 3))
0
>(profundidade '(1 2 (3 ((4) (5 (6))))))
4
```

Exercício 1.24
Defina a função `conta-todos` que recebe um elemento e uma lista e devolve o número de vezes que o elemento ocorre na lista e nas suas sublistas.

```
>(conta-todos 5 '(1 2 3))
0
>(conta-todos 3 '(1 2 3))
1
>(conta-todos 2 '(1 ((2) 3 2) ((1 2))))
3
```

Exercício 1.25
Defina a função `mapeia` que recebe uma função de um argumento e uma lista e devolve uma lista que contém os resultados de aplicar a função a cada um dos elementos da lista.

```
>(mapeia #'1+ '(1 2 3))
(2 3 4)
```

Exercício 1.26
Defina a função `remove-se` que recebe uma função de teste (de um argumento) e uma lista e retorna a lista que corresponde à lista argumento à qual foram retirados todos os elementos que verificam a função de teste.

```
>(remove-se #'(lambda (n) (> n 5)) '(1 2 3 4 5 6 7 8 9))
(1 2 3 4 5)
```

Exercício 1.27
Defina a função `todos?` que recebe uma lista de elementos e uma função de teste (de uma argumento) e retorna verdade se todos os elementos da lista verificam a função e falso em caso contrário.

```
>(todos? '(1 2 3 4 5 6 7 8 9) #'(lambda (n) (> n 5)))
nil
>(todos? '(1 2 3 4) #'(lambda (n) (< n 5)))
t
```

Exercício 1.28
Defina a função `reduza` que recebe uma função e uma lista. A lista deve ter pelo menos dois elementos. A função é chamada inicialmente para os dois primeiros elementos da lista, depois entre esse resultado e o elemento seguinte da lista e assim sucessivamente.

```
>(reduza #'+ '(1 2 3 4))
10
>(reduza #'* '(1 2 3 4))
24
>(reduza #'interseccao '((b l a d) (b a d) (r a t)))
(A)
```

Exercício 1.29
Defina a função `soma-matrizes` que recebe duas tabela de 2 dimensões (x, y) de igual tamanho arbitrário e retorna outra tabela que corresponde à soma das duas tabelas.

Exercício 1.30
Defina a função `transpoe` que recebe uma tabela de 2 dimensões (x, y) de tamanho arbitrário e retorna outra tabela que corresponde à transposição da tabela recebida em argumento.

Exercício 1.31
Defina a função `diagonal` que recebe uma tabela de 2 dimensões (x, y) de tamanho arbitrário e retorna verdade se esta é uma tabela diagonal e falso em caso contrário.

Exercício 1.32
Defina a função `copy-array-2-dim` que recebe uma tabela bidimensional de tamanho arbitrário e retorna uma cópia dessa tabela.

Exercício 1.33
Usando estruturas:

1. Defina uma estrutura que representa informação sobre um modelo de carro. Considere que um modelo é caracterizado por uma marca, designação, potência, cilindrada e extras.
2. Defina uma estrutura veículo que representa informação sobre um veículo. Considere que este é caracterizado por uma matrícula, ano de registo e um modelo de carro.
3. Crie um veículo de matrícula 12-34-AB, ano de registo 1900, marca Rolls-Royce, designação Silver-Ghost, 300 Cv de potência, 20000cm^3 de cilindrada e sem extras.
4. Escreva uma função `veiculo-cilindrada` que recebe um veículo e retorna a cilindrada desse veículo.

Exercício 1.34
Considere o clássico jogo do galo, num tabuleiro de 3x3, em que há dois jogadores representados por X e por O respectivamente.

1. Defina uma estrutura `estado` que contenha a seguinte informação:
   (a) Um campo `tabuleiro`: array de 3x3 que representa o tabuleiro do jogo. Cada posição conterá o símbolo X, O ou NIL, consoante tenha uma peça do jogador respectivo ou nenhuma peça.
   (b) Um campo `proximo`: irá conter o símbolo X ou O consoante o próximo jogador a jogar.
2. Defina a função `cria-estado` que devolve uma instância da estrutura do tipo `estado`, devidamente inicializada, em que o primeiro jogador é o X.
3. Considere a seguinte função:

```
> (defun joga (est x y)
    (cond ((or (< x 0) (< y 0) (> x 2) (> y 2)) nil)
          (T (setf (aref (estado-tabuleiro est) x y)
                   (estado-proximo est)
                   (estado-proximo est)
                   (if (eql (estado-proximo est) 'X)
                      'O
                      'X))
             est)))
```

   (a) Diga qual o resultado obtido pela seguinte expressão:

```
> (let ((est (cria-estado)))
       (let ((est2 (joga est 1 1)))
           est2))
```

   (b) Obteria o mesmo resultado com a expressão seguinte? Porquê?

```
> (let ((est (cria-estado))
        (est2 (joga est 1 1)))
       est2)
```

4. Imagine agora que pretente definir a função `joga` de forma não-destrutiva, isto é, após executar est2, esta deverá ficar com o novo estado e est com o estado original, não alterado. A função seguinte resolve o nosso problema? Caso esta função não resolva o nosso problema diga o que falhou e como o poderíamos resolver?

```
(defun joga (est x y)
  (cond ((or (< x 0) (< y 0) (> x 2) (> y 2)) nil)
        (T (let ((novo-est (copy-estado est)))
             (setf (aref (estado-tabuleiro novo-est) x y)
                   (estado-proximo novo-est)
                   (estado-proximo novo-est)
                   (if (eql (estado-proximo novo-est) 'X)
                      'O
                      'X))
             novo-est))))
```

Exercício 1.35
Considere a seguinte função:

```
> (defun misterio (n1)
     #'(lambda (x n2)
          (cond ((eql x 'd) (setf n1 (+ n1 n2)))
                ((and (eql x 'l) (<= n2 n1)) (setf n1 (- n1 n2)))
                ((and (eql x 'l) (> n2 n1)) 'nao-tem-suficiente)
                (t 'operacao-desconhecida))))
MISTERIO
```

1. Explique o que faz a função mistério e dê exemplos da sua utilização.

2. Explique o que acontece em cada passo se as seguintes expressões Lisp forem avaliadas depois da definição da função mistério. Diga qual o valor retornado em cada passo dessa sequência.

```
> (setf minha (misterio 200))

> (funcall minha 'l 100)

> (funcall minha 'lav 0)

> (funcall minha 'l 0)

> (funcall minha 'l 200)

> (funcall minha 'd 100)

> (funcall minha 'd 0)

> (funcall minha 'd 200)
```

Exercício 1.36
Defina a função `eleva`, que eleva um número a uma determinada potência. Se a potência não for indicada deverá ser considerada 2.

```
> (eleva 3)
9
> (eleva 3 4)
81
```

Exercício 1.37
Defina uma variação da função anterior, `eleva*`, que eleva um número a uma determinada potência em que deixa de ser necessário saber a ordem dos seus argumentos.

```
> (eleva* :base 3 :expoente 4)
81
```

Exercício 1.38
Defina a função `faz-lista` que recebe qualquer número de argumentos e devolve uma lista com todos eles.

```
> (faz-lista 1 2 3 4)
(1 2 3 4)
```