* PROBLEMA DA MULTIPLICAÇÃO DE CADEIA DE MATRIZES * (Cormen, §15.2)

0. ENTRADA: um vetor d[0..n] com as dimensões de "n" matrizes a serem
            multiplicadas, A_1 ... A_n, sendo A_i uma matriz d[i-1] x d[i].
   SAÍDA: o menor número de multiplicações numéricas (ou "escalares")
          necessárias para se realizar a multiplicação das "n" matrizes.

1. EXERCÍCIO: Verifique se o seguinte procedimento sempre leva a uma solução
   ótima:

   1) Faça D := máx { d[i] : 0 < i < n }.

   2) Coloque parênteses ao redor de cada par A_i*A_{i+1} tal que A tenha D
      colunas e B tenha D linhas.

   3) Substitua cada par A_i*A_{i+1} parentetizado no passo anterior por uma
      única matriz B_i, de dimensões d[i-1] x d[i+1].

   4) Resolva a instância resultante recursivamente.

Observação: os passos acima levam a uma parentetização das "n" matrizes; obter o número de multiplicações numéricas utilizado nessa parentetização é uma tarefa menos importante perto da questão de se essa parentetização é ótima ou não.

2. Um algoritmo de divisão-e-conquista:

```
============================================================
Algoritmo: cad_matrizes
Entrada: vetor d[0..n] de inteiros.
Saída: inteiro.
------------------------------------------------------------
1. SE n = 1
2. | RETORNE 0.
3. mult_mín := INFINITO
4. PARA i DE 1 A n-1
5. | mult_i := d[0] * d[i] * d[n]
   |           + cad_matrizes( i, d[0..i] )
   |           + cad_matrizes( n-i, d[i..n] )
6. | SE mult_i < mult_mín
7. | | mult_mín := mult_i
8. RETORNE mult_mín.
============================================================
```

3. EXERCÍCIO: prove ou refute: o algoritmo acima executa em tempo Ω(2^n).

4. Uma versão memoizada do algoritmo acima:

```
============================================================
Algoritmo: cad_matrizes_memo
Entrada: vetor d[0..n] de inteiros.
Saída: inteiro.
------------------------------------------------------------
1. Obter uma matriz M[1..n][1..n] de inteiros
2. PARA i DE 1 A n
3. | M[i][i] := 0
4. | PARA j DE i+1 A n
5. | | M[i][j] := -1
6. RETORNE cad_matrizes_rec(d, M, 1, n).
============================================================
```

```
============================================================
Algoritmo: cad_matrizes_rec
Entrada: vetor d[0..n] de inteiros,
         matriz M[1..n][1..n] de inteiros,
         inteiros "i" e "j".
Saída: inteiro.
------------------------------------------------------------
1. SE M[i][j] = -1
2. | mult_mín := INFINITO
3. | PARA k DE i A j-1
4. | | mult_k := d[i-1] * d[k] * d[j]
   | |           + cad_matrizes_rec( d, M, i,   k)
   | |           + cad_matrizes_rec( d, M, k+1, j)
5. | | SE mult_k < mult_mín
6. | | | mult_mín := mult_k
7. | M[i][j] := mult_mín
8. RETORNE M[i][j].
============================================================
```

5. EXERCÍCIO:

   a) Escreva uma versão não-recursiva do algoritmo acima.

   b) Argumente que essa versão executa em tempo O(n^3).

   c) Ela executa também em tempo Ω(n^3)? (Bom exercício.)

   d) Amplie o algoritmo iterativo em questão, de forma que ele compute não apenas o número de multiplicações numéricas da parentetização ótima, mas também a parentetização em si. DICA: retorne uma matriz K[1..n][1..n] tal que, para toda subcadeia A_i ... A_j, a parentetização ótima é (A_i ... A_k) (A_{k+1} ... A_j), onde k = K[i][j].

   e) Modifique o algoritmo anterior, de forma que ele passe a receber como entrada também as matrizes A_1 ... A_n em si, e de forma que a saída do algoritmo seja apenas a matriz produto final, obtida a partir da utilização do algoritmo padrão de multiplicação de matrizes segundo a ordem de multiplicação de uma parentetização ótima.