Exercício 1

Uma expressão completamente parentizada ou é uma matriz M, ou é da forma (AB), onde A e B são expressões completamente parentizadas.

Vamos provar por indução que o número de pares de parênteses numa expressão completamente parentizada com n matrizes é $n-1$. Se $n=1$, a expressão tem só uma matriz e é portanto da forma M, contendo 0 pares de parênteses.

Considere agora uma expressão completamente parentizada com $n>1$ matrizes. Então a expressão é da forma (AB), onde ambas A e B são expressões completamente parentizadas. Suponha que A contenha n_1 matrizes e B contenha n_2 matrizes, sendo $n_1+n_2=n$.

Como n_1, $n_2 < n$, temos por hipótese de indução que o número de pares de parênteses em A e B é n_1-1 e n_2-1, respectivamente. Assim, o número de pares de parênteses em (AB) é

$$(n_1-1)+(n_2-1)+1 = n_1+n_2-1 = n-1,$$

como queríamos.

Exercício 2

Suponha que $s \neq t$. Seja $v \in V$ tal que $(v,t) \in A$. Então se $s=v_1,\ldots,v_k=v$ é um (s,v)-caminho simples em D, temos que $s=v_1,\ldots,v_k,t$ é um (s,t)-caminho simples. Isso é verdade pois D é acíclico, logo t não pode ser um dos vértices v_1, $\ldots$, v_k.

Seja $c(v)$ o número de (s,v)-caminhos simples em D. Se $s=v$, temos $c(v)=1$. Caso contrário temos

$$c(v) = \sum_{u:(u,v)\in A} c(u). \tag{1}$$

Um algoritmo para calcular o número de (s,t)-caminhos simples é o seguinte:

(1) Seja v_1, $\ldots$, v_n uma ordenação topológica de D. Suponha, sem perda de generalidade, que $s=v_1$ e $t=v_n$.
(2) Para $i=1,\ldots,n$, calcule $c(v_i)$ usando (1).

Para calcular $c(v)$, precisamos conhecer $c(u)$ para todo $(u,v)\in A$. Por isso, como usamos uma ordenação topológica, o passo (2) está bem definido.

Para executar o passo (1) gastamos tempo $O(|V|+|A|)$. Para executar o passo (2) precisamos percorrer todos os vértices do grafo e para executar as somas olhamos para cada arco exatamente uma vez. Assim, gastamos para o passo (2) tempo $O(|V|+|A|)$.

Exercício 3

Considere o grafo $D=(V,A)$ com $V=1,2,3$ e $A=\{(1,2),(1,3),(2,3)\}$. Considere a função comprimento l tal que $l(1,2)=2$, $l(1,3)=1$ e $l(2,3)=-2$.

Quando o algoritmo de Dijkstra é executado a partir do vértice 1, ele encontrará distâncias $d_1=0$, $d_2=2$ e $d_3=1$, ao passo que o caminho mínimo entre 1 e 3 tem comprimento 0.

Exercício 4

Seja $x\in\Sigma^*$ uma palavra de comprimento n. Note que, se $x\in L^*$, então $x\in L^0\cup L^1\cup\cdots\cup L^n$. De fato, suponha que $x\in L^m$ com $m>n$. Então

$$x = x_1x_2\cdots x_m$$

com x_1, $\ldots$, $x_m\in L$. Mas, como x tem comprimento n, pelo menos $m-n$ das palavras x_i devem ser a palavra vazia. Mas então, eliminando essas palavras, vemos que $x\in L^0\cup L^1\cup\cdots\cup L^n$, como queríamos.

Defina:

$$c(i,k) = \begin{cases} 1 & \text{se } x[1\ldots i] \in L^k, \\ 0 & \text{caso contrário,} \end{cases}$$

para $i = 0, \ldots, n$ e $k = 0, \ldots, n$.

Podemos considerar os seguintes casos:

(1) se $k = 0$, então $c(i,k) = 1$ se e somente se $i = 0$;
(2) se $k > 0$, então $c(i,k) = 1$ se e somente se existe $j \leq i$ tal que $x[1\ldots j] \in L^{k-1}$ e $x[j+1\ldots i] \in L$.

Assim chegamos à seguinte fórmula para c:

$$c(i,k) = \begin{cases} 1 & \text{se } k = 0 \text{ e } i = 0, \\ 0 & \text{se } k = 0 \text{ e } i > 0, \\ \bigvee_{j=0}^{i} c(j,k-1) \wedge (x[j+1\ldots i] \in L) & \text{caso contrário.} \end{cases}$$

Usando essa fórmula, podemos desenvolver um algoritmo que recebe uma palavra $x \in \Sigma^*$ e devolve 1 se $x \in L^*$ e 0 caso contrário.

```
Testa-Palavra(x)
    n = |x|
    c(0,0) = 1
    para i = 1, ..., n: c(i,0) = 0
    para k = 1, ..., n:
        para i = 0, ..., n:
            c(i,k) = 0
            para j = 0, ..., i:
                c(i,k) = c(i,k) ∨ (c(j,k-1) ∧ (x[j+1...i] ∈ L))
    para k = 0, ..., n:
        se c(n,k) = 1: devolva 1
    devolva 0
```

Qual o tempo gasto pelo algoritmo? Suponha que o algoritmo que decide se uma palavra x de comprimento n pertence à L consome tempo $O(n^k)$. Então o algoritmo acima executa esse algoritmo $O(n^3)$ vezes em palavras de comprimento no máximo n. Assim, o tempo total gasto é $O(n^{k+3})$ e temos portanto um algoritmo polinomial.