1º Teste de Análise e Desenho de Algoritmos

Departamento de Informática, FCT NOVA
29 de Abril de 2019
Duração: 1 hora e 45 minutos

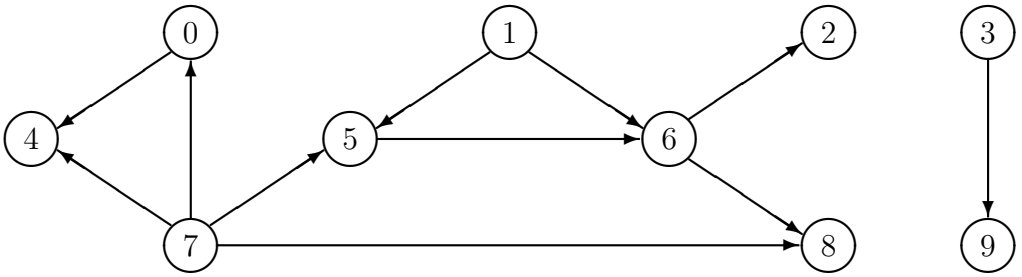
Tem de entregar os 2 cadernos, um com 2 folhas e o outro com 3.

Os cadernos não podem ser desagrafados.

Identifique os cadernos com o seu número e o seu nome.

Número: _____ **Nome:** _____

Pergunta 1 Suponha que se executa o algoritmo *topologicalSort* com o grafo G esquematizado na figura, depois de se ter alterado a disciplina do saco *ready* para *last-in first-out* (ou seja, quando *ready* é uma pilha).



Assuma que os métodos *nodes* e *outAdjacentNodes* iteram sempre os vértices por ordem crescente. Por exemplo, $G.outAdjacentNodes(7)$ produz os vértices 0, 4, 5 e 8 (por esta ordem).

(a) [2.5 valores] Qual seria a permutação de vértices computada (o conteúdo do vetor retornado)?

Permutação:

7	0	4	3	9	1	5	6	8	2
0	1	2	3	4	5	6	7	8	9

(b) [0.5 valores] Indique o maior número de vértices presentes simultaneamente na pilha (o maior valor de *ready.size*() durante a execução do algoritmo).

Tamanho máximo da pilha: 3

Pergunta 2 Considere a seguinte função recursiva, $d_{X,Y}(i)$, onde:

- $X = (x_0, x_1, \ldots, x_n)$ e $Y = (y_0, y_1, \ldots, y_n)$ são duas sequências de inteiros (com $n \geq 0$);
- i é um inteiro entre 0 e $n+1$ $(0 \leq i \leq n+1)$.

$$d_{X,Y}(i) = \begin{cases} 0, & \text{se } i = 0; \\ |x_0 - y_0|, & \text{se } i = 1; \\ \min(\ d_{X,Y}(i-1) + |x_{i-1} - y_{i-1}|, \\ \qquad d_{X,Y}(i-2) + |x_{i-1} - y_{i-2}| + |x_{i-2} - y_{i-1}|\ ), & \text{se } i \geq 2. \end{cases}$$

Note que as sequências X e Y não variam entre chamadas recursivas. Por esse motivo, optou-se por escrever $d_{X,Y}(i)$ em vez de $d(X, Y, i)$.

(a) [5 valores] Apresente um algoritmo iterativo, desenhado segundo a técnica da programação dinâmica e implementado em Java, que receba duas sequências de inteiros (guardadas em vetores do tipo `int[]`):

$$X = (x_0, x_1, \ldots, x_n) \text{ e } Y = (y_0, y_1, \ldots, y_n) \qquad (\text{com } n \geq 0)$$

e calcule o valor de $d_{X,Y}(n+1)$.

```
int funD( int[] x, int[] y ) {
   int[] d = new int[x.length + 1];
   // d[0] = 0;
   d[1] = Math.abs(x[0] - y[0]);
   for ( int i = 2; i < d.length; i++ ) {
      d[i] = d[i-1] + Math.abs(x[i-1] - y[i-1]);
      int v = d[i-2] + Math.abs(x[i-1] - y[i-2]) + Math.abs(x[i-2] - y[i-1]);
      if ( v < d[i] )
         d[i] = v;
   }
   return d[x.length];
}
```

(b) [0.5 valores] Qual é a complexidade espacial do seu algoritmo? Justifique a sua resposta.

A complexidade espacial é $\Theta(k)$, onde k é o comprimento dos vetores de entrada, porque o vetor `d` tem comprimento $k + 1$.

(c) [0.5 valores] Qual é a complexidade temporal do seu algoritmo? Justifique a sua resposta.

A complexidade temporal é $\Theta(k)$, onde k é o comprimento dos vetores de entrada, porque o ciclo *for* tem $\Theta(k)$ passos e cada passo é constante.

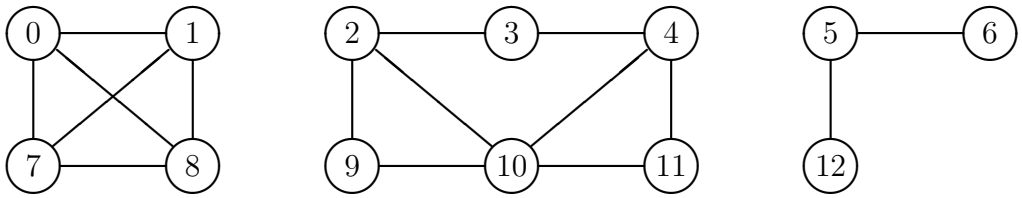
NOTA MUITO IMPORTANTE:
O que escrever nesta página não será avaliado.

Pergunta 3 O estudo das redes sociais permite compreender e prever comportamentos. Neste exercício, pretende-se calcular a coesão média dos grupos de indivíduos.

Uma rede social é modelada por um grafo não orientado. Cada vértice representa um indivíduo e a existência de um arco entre dois vértices indica que os dois indivíduos são amigos. Com base nesta relação de amizade podem-se definir caminhos (da forma usual em grafos) e componentes conexas. O *coeficiente de coesão* de uma componente conexa mede "a densidade" das amizades. Se C for uma componente conexa com $m \geq 2$ vértices e n arcos:

$$\text{coef-coesão}(C) = \frac{2\,n}{m\,(m-1)}.$$

O *coeficiente médio de coesão* de uma rede social é a média dos coeficientes de coesão das componentes conexas da rede que têm pelo menos dois vértices.



Para exemplificar, considere a rede social esquematizada na figura acima, que tem três componentes conexas.

- A componente conexa da esquerda, denotada por $C_1 = (V_1, A_1)$, tem 4 vértices e 6 arcos:

  $$V_1 = \{0, 1, 7, 8\} \text{ e } A_1 = \{(0,1), (0,7), (0,8), (1,7), (1,8), (7,8)\}.$$

  Portanto, o seu coeficiente de coesão é $\text{coef-coesão}(C_1) = \dfrac{2 \times 6}{4 \times 3} = 1$.

- A componente conexa do centro, denotada por $C_2 = (V_2, A_2)$, tem 6 vértices e 8 arcos:

  $$V_2 = \{2, 3, 4, 9, 10, 11\} \text{ e } A_2 = \{(2,3), (2,9), (2,10), (3,4), (4,10), (4,11), (9,10), (10,11)\}.$$

  Logo, $\text{coef-coesão}(C_2) = \dfrac{2 \times 8}{6 \times 5} \approx 0.533$.

- A componente conexa da direita, denotada por $C_3 = (V_3, A_3)$, tem 3 vértices e 2 arcos:

  $$V_3 = \{5, 6, 12\} \text{ e } A_3 = \{(5,6), (5,12)\}.$$

  Aplicando a fórmula, $\text{coef-coesão}(C_3) = \dfrac{2 \times 2}{3 \times 2} \approx 0.667$.

O coeficiente médio de coesão da rede é a média dos três coeficientes de coesão:

$$\frac{\text{coef-coesão}(C_1) + \text{coef-coesão}(C_2) + \text{coef-coesão}(C_3)}{3} \approx \frac{1 + 0.533 + 0.667}{3} \approx 0.733.$$

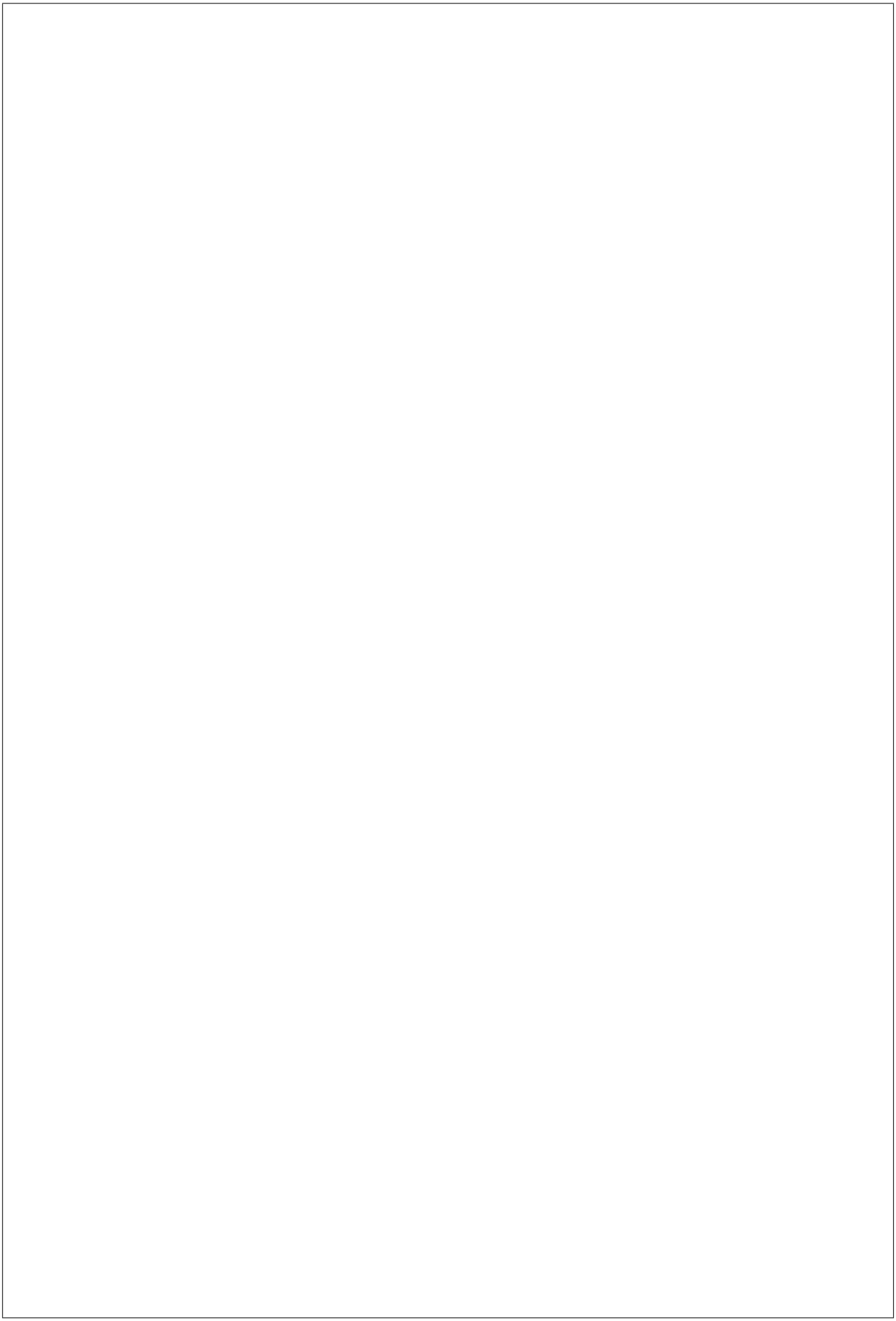
Pretende-se uma função que receba uma rede $R = (V, A)$, que é um grafo não orientado, e que calcule o coeficiente médio de coesão da rede. Assuma que todos os indivíduos têm algum amigo.

Número: _____ **Nome:** _____

(a) [5 valores] Apresente a função pretendida (em pseudo-código).

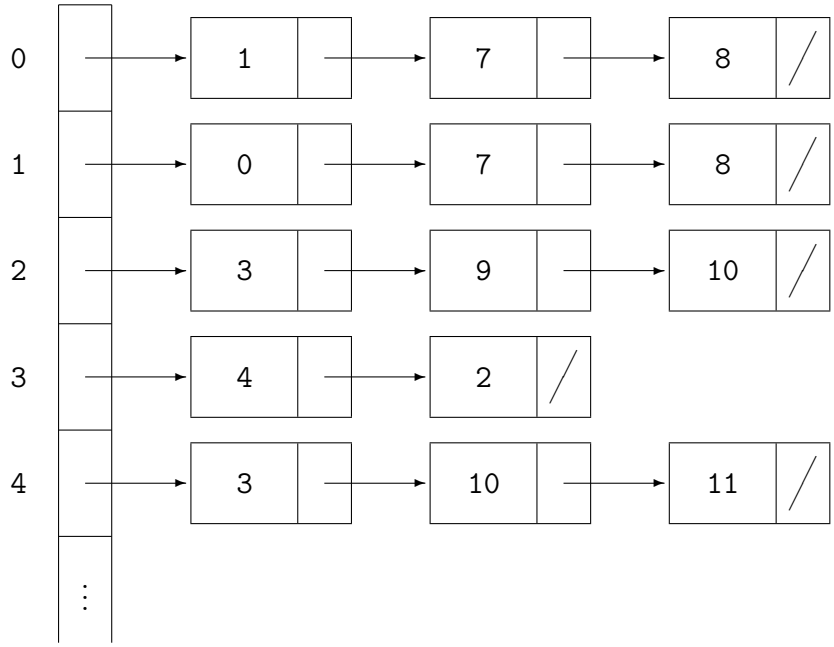
```
double meanCohesionCoeff( UndiGraph graph ) {
    boolean[] found = new boolean[ graph.numNodes() ];
    for every Node v in graph.nodes()
        found[v] = false;
    double sum = 0.0;
    int nCC = 0;
    for every Node v in graph.nodes()
        if ( !found[v] ) {
            sum += cohesionCoeff(graph, found, v);
            nCC++;
        }
    return sum / nCC;
}


double cohesionCoeff( UndiGraph graph, boolean[] found, Node root ) {
    Queue<Node> waiting = new QueueIn...<>(?);
    int nV = 0;
    int nE = 0;
    waiting.enqueue(root);
    found[root] = true;
    do {
            Node node = waiting.dequeue();
            nV++;
            nE += node.degree();
            for every Node v in graph.adjacentNodes(node)
                if ( !found[v] ) {
                    waiting.enqueue(v);
                    found[v] = true;
                }
        }
    while ( !waiting.isEmpty() );
    return (1.0 * nE) / (nV * (nV − 1));
}
```



(b) [1 valor] Que estruturas de dados escolheria para implementar o grafo? Não escreva código, mas ilustre a sua resposta com o grafo do exemplo. Como o grafo é grande, ilustre apenas com uma parte.

Guardaria o grafo num vetor de listas ligadas de adjacências.



(c) [1 valor] Qual é a complexidade temporal do seu algoritmo, no pior caso, assumindo que o grafo está implementado como indicou na alínea anterior? Justifique a sua resposta.

A complexidade temporal é $\Theta(|V| + |A|)$, onde (V, A) é o grafo de entrada, porque o algoritmo efetua um percurso em largura (num grafo implementado com um vetor de listas ligadas de adjacências) e as instruções adicionais para calcular o coeficiente médio de coesão têm todas complexidade constante. Em particular, o grau de um vértice é o tamanho da sua lista de adjacências.

Pergunta 4 O Carlos, que começou a estudar Lógica, já sabe que é necessário colocar parêntesis numa expressão booleana para definir a ordem pela qual as operações são efetuadas, porque ordens diferentes podem conduzir a resultados diferentes.

Por exemplo, a expressão `T xor T and F or F` é ambígua (onde `T` denota *true* e `F` representa *false*). Quando se colocam os três pares de parêntesis para definir a ordem das três operações, obtém-se uma das cinco *expressões parentisadas* da tabela abaixo. Três dessas expressões são verdadeiras e duas são falsas.

Expressão Parentisada	Valor da Expressão
`(((T xor T) and F) or F)`	`F`
`((T xor (T and F)) or F)`	`T`
`((T xor T) and (F or F))`	`F`
`(T xor (T and (F or F)))`	`T`
`(T xor ((T and F) or F))`	`T`

Pretende-se definir **uma função matemática recursiva** que, com base:

- num inteiro positivo, k, que representa o número de operações, e
- numa expressão booleana sem parêntesis:

$$E \;=\; v_0\; o_1\; v_1\; o_2\; v_2\; o_3\; v_3\; \cdots\; v_{k-1}\; o_k\; v_k$$

onde $v_i \in \{\texttt{F}, \texttt{T}\}$ e $o_j \in \{\texttt{or}, \texttt{xor}, \texttt{and}\}$, para $i = 0, \ldots, k$ e $j = 1, \ldots, k$,

calcula o número m de expressões parentisadas de E cujo valor é `T` e o número n de expressões parentisadas de E cujo valor é `F`. O resultado será o par (m, n).

Para o nosso exemplo, em que $k = 3$ e $E =$ `T xor T and F or F`, o resultado seria $(3, 2)$.

(a) [3.5 valores] Defina a função pretendida e indique claramente o que representa cada uma das variáveis que utilizar.

$\mathcal{C}_E(i, j)$ é o par pretendido quando a expressão sem parêntesis é a seguinte parte de E:

$$v_i\; o_{i+1}\; v_{i+1}\; o_{i+2}\; v_{i+2}\; \cdots\; v_{j-1}\; o_j\; v_j$$

$$\mathcal{C}_E(i,j) = \begin{cases} (0,1) & \text{se } i = j \text{ e } v_i = \texttt{F} \\ (1,0) & \text{se } i = j \text{ e } v_i = \texttt{T} \\ \displaystyle\sum_{k=i+1}^{j} \mathcal{L}(o_k,\, \mathcal{C}_E(i, k-1),\, \mathcal{C}_E(k, j)) & \text{se } i < j \end{cases}$$

$\mathcal{L}$ é a seguinte função auxiliar:

$$\mathcal{L}(o,\, (a,b),\, (x,y)) = \begin{cases} (ax,\; ay + bx + by) & \text{se } o = \texttt{and} \\ (ax + ay + bx,\; by) & \text{se } o = \texttt{or} \\ (ay + bx,\; ax + by) & \text{se } o = \texttt{xor} \end{cases}$$

(b) [0.5 valores] Indique a chamada inicial (a chamada que resolve o problema).

$\mathcal{C}_E(0, k)$