

Exame de Análise e Desenho de Algoritmos

Departamento de Informática, FCT NOVA

20 de Junho de 2020

Duração: 3 horas

Caderno 1 (3 folhas e 3 perguntas)

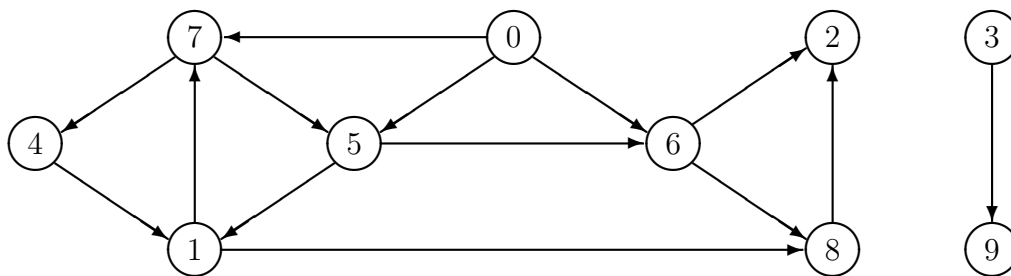
Tem de entregar os 2 cadernos.

Os cadernos não podem ser desagradados.

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

Número: _____ Nome: _____

Pergunta 1 Suponha que se executa a **versão iterativa** do algoritmo *dfsTraversal* (que percorre os vértices de um grafo em profundidade) com o grafo G esquematizado na figura.



Assuma que os métodos *nodes* e *outAdjacentNodes* iteram sempre os vértices por ordem crescente. Por exemplo, $G.outAdjacentNodes(0)$ produz os vértices 5, 6 e 7 (por esta ordem).

Atenção: A implementação do método *dfsExplore* é **iterativa** (não há chamadas recursivas).

- (a) [2 valores] Indique a ordem pela qual os vértices são processados. Um vértice v é processado quando é executada a atribuição `processed[v] = true`.

--

- (b) [0.5 valores] Quantas árvores são geradas (quantas vezes o método *dfsExplore* é executado)?

--

- (c) [0.5 valores] Quantos níveis tem a maior árvore gerada? Note que a raiz está no primeiro nível, os filhos da raiz estão no segundo nível, os netos da raiz estão no terceiro nível, etc.

--

Pergunta 2 Considere a seguinte função recursiva, $f_{S,k}(i, j)$, onde:

- $S = (x_0, x_1, \dots, x_{n-1})$ é uma sequência de números inteiros (com $n \geq 1$);
- k é um número inteiro;
- i e j são dois números inteiros entre 0 e $n - 1$ ($0 \leq i < n$ e $0 \leq j < n$).

$$f_{S,k}(i, j) = \begin{cases} x_j, & \text{se } i = 0 \text{ e } 0 \leq j < n; \\ x_i, & \text{se } j = 0 \text{ e } 1 \leq i < n; \\ f_{S,k}(i-1, j-1), & \text{se } i \geq 1 \text{ e } j \geq 1 \text{ e } f_{S,k}(i, j-1) = f_{S,k}(i-1, j); \\ k + f_{S,k}(i-1, j-1), & \text{se } i \geq 1 \text{ e } j \geq 1 \text{ e } f_{S,k}(i, j-1) \neq f_{S,k}(i-1, j). \end{cases}$$

Note que a sequência S e o número k não variam entre chamadas recursivas. Por esse motivo, optou-se por escrever $f_{S,k}(i, j)$ em vez de $f(S, k, i, j)$.

- (a) [2.5 valores] Apresente um algoritmo iterativo, desenhado segundo a técnica da programação dinâmica e implementado em Java, que receba uma sequência de n números inteiros (guardada num vetor do tipo `int[]` com comprimento n) e um número inteiro:

$$S = (x_0, x_1, \dots, x_{n-1}) \text{ e } k \quad (\text{com } n \geq 1)$$

e calcule o valor de $f_{S,k}(n-1, n-1)$.

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

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

Pergunta 3 Neste exercício pretende-se implementar o algoritmo de Dijkstra em Java, usando a classe *PriorityQueue*, disponível na biblioteca. Uma *PriorityQueue* é uma fila com prioridade implementada em *heap* binário (que não oferece a operação *decreaseKey*).

- (a) [1.5 valores] Implemente o método *dijkstra*, completando os corpos dos dois ciclos assinalados na página seguinte. Considere que, para além dos métodos *isEmpty* e *size*, só há mais três métodos oferecidos por uma *PriorityQueue* (de elementos do tipo E):

- **boolean** *add*(E e); // Inserts the specified element in this queue.
- E *remove*() **throws** NoSuchElementException;
// Retrieves and removes the least element from this queue (ties are broken arbitrarily).
- **boolean** *remove*(Object o);
// Removes a single instance of the specified element from this queue, if it is present.

```
class Edge {
    private int destin , weight; // Destination node and weight

    public Edge( int destination , int theWeight ) {
        destin = destination;
        weight = theWeight;
    }

    public int getDestin( ) { return destin; }

    public int getWeight( ) { return weight; }
}

class Entry implements Comparable<Entry> {
    private int key, val;

    public Entry( int theKey, int value ) {
        key = theKey;
        val = value;
    }

    public int getKey( ) { return key; }

    public int getValue( ) { return val; }

    public void setKey( int newKey ) { key = newKey; }

    public void setValue( int newValue ) { val = newValue; }

    public boolean equals( Object object ) {
        if ( object instanceof Entry ) {
            Entry entry = (Entry) object;
            return key == entry.key && val == entry.val;
        }
        return false;
    }

    public int compareTo( Entry entry ) {
        int compResult = key - entry.key;
        if ( compResult != 0 )
            return compResult;
        return val - entry.val;
    }
}
```

```

import java.util.List;      import java.util.LinkedList;
import java.util.Queue;    import java.util.PriorityQueue;

public class Problem {
    private int numNodes;      // Number of nodes
    private List<Edge>[] edges; // The adjacency linked lists

    @SuppressWarnings("unchecked")
    public Problem( int numberOfNodes ) {
        numNodes = numberOfNodes;
        edges = new List[numNodes];
        for ( int i = 0; i < numNodes; i++ )
            edges[i] = new LinkedList<>();
    }

    public void addEdge( int tail, int head, int dist ) {
        edges[tail].add( new Edge(head, dist) );
    }

    public int[] dijkstra( int origin ) {
        boolean[] selected = new boolean[numNodes];
        int[] length = new int[numNodes];
        Queue<Entry> connected = new PriorityQueue<>(); // Capacidade omitida
        for ( int i = 0; i < numNodes; i++ )           // propositadamente.
            length[i] = Integer.MAX_VALUE;
        length[origin] = 0;
        connected.add( new Entry(0, origin) );
        do {
            int node = connected.remove().getValue();
            // Complete o corpo deste ciclo

        }
        while ( !connected.isEmpty() );
        return length;
    }

    private void exploreNode( int source, boolean[] selected, int[] length,
                             Queue<Entry> connected ) {
        for ( Edge e : edges[source] ) {
            int node = e.getDestin();
            // Complete o corpo deste ciclo

        }
    }
}

```

- (b) [1 valor] Qual é a complexidade temporal do seu método *dijkstra*? Justifique a sua resposta, denotando por (V, A) o grafo guardado nas variáveis *numNodes* e *edges*.

- (c) [1 valor] Qual é a complexidade espacial do seu método *dijkstra*? Justifique a sua resposta, denotando por (V, A) o grafo guardado nas variáveis *numNodes* e *edges*.

Exame de Análise e Desenho de Algoritmos

Departamento de Informática, FCT NOVA

20 de Junho de 2020

Duração: 3 horas

Caderno 2 (4 folhas e 3 perguntas)

Tem de entregar os 2 cadernos.

Os cadernos não podem ser desagradados.

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

Número: _____ Nome: _____

Pergunta 4 A classe *SlackStack* implementa filas com disciplina LIFO, de elementos do tipo E, com uma pilha implementada em vetor. Essa pilha (chamada `lifo`) tem o dobro da capacidade indicada no construtor da *SlackStack*. Quando se empilha um elemento numa *SlackStack* e a sua `lifo` está cheia, remove-se metade dos elementos da `lifo` (a metade mais antiga), fazendo uso da pilha `aux`.

```
import dataStructures.Stack;
import dataStructures.StackInArray;

public class SlackStack<E> {

    private Stack<E> lifo;
    private Stack<E> aux;
    private int trueCapacity;

    public SlackStack( int capacity ) {
        trueCapacity = 2 * capacity;
        lifo = new StackInArray<>(trueCapacity);
        aux = new StackInArray<>(capacity);
    }

    public void push( E elem ) {
        if ( lifo.size() == trueCapacity )
            this.deleteOldHalf();
        lifo.push(elem);
    }

    private void deleteOldHalf( ) {
        int half = trueCapacity / 2;
        for ( int i = 0; i < half; i++ )
            aux.push( lifo.pop() );
        for ( int i = 0; i < half; i++ )
            lifo.pop();
        for ( int i = 0; i < half; i++ )
            lifo.push( aux.pop() );
    }

    public E pop( ) throws RuntimeException {
        if ( lifo.size() == 0 )
            throw new RuntimeException("The_stack_is_empty");

        return lifo.pop();
    }

}
```

Considere a função $\Phi(S)$, que atribui a cada objeto S da classe *SlackStack* o dobro do número de elementos guardados na pilha $S.lifo$:

$$\Phi(S) = 2 \times S.lifo.size().$$

- (a) [0.6 valores] Prove que Φ é uma função potencial válida.

- (b) [2.9 valores] Calcule as complexidades amortizadas dos métodos *push* e *pop* da classe *SlackStack*, justificando. Assuma que os métodos *size*, *push* e *pop* da classe *StackInArray* têm complexidade constante. No estudo da complexidade amortizada do método *push*, analise separadamente os casos em que a condição do **if** é: falsa; verdadeira. No estudo da complexidade amortizada do método *pop*, assumo que não é levantada a exceção.

Recorde que não se usam constantes multiplicativas nos custos reais. Por exemplo, se um custo real c for de ordem $2n$, onde n é uma variável que denota uma quantidade positiva, $c = n$.

Pergunta 5 A paciência MINREVERSES joga-se invertendo arcos num grafo, tendo como objetivo “construir” um caminho entre dois vértices com o menor número possível de inversões.

No início do jogo, o jogador recebe um grafo e dois vértices distintos, x e y . O grafo é orientado, não pesado e fracamente conexo (há caminho entre dois quaisquer vértices, se se ignorar o sentido dos arcos). Em cada jogada, o jogador altera o grafo, invertendo o sentido de um dos arcos. Em qualquer momento, o jogador pode terminar o jogo, assinalando que não quer fazer mais alterações. Considera-se que o jogo foi ganho se há algum caminho de x para y no último grafo obtido e o número de jogadas foi o menor possível. A ordem das jogadas é irrelevante.

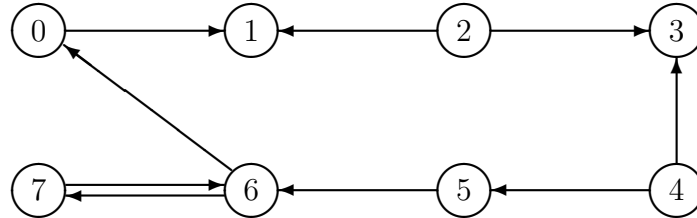


Figura 1: Grafo $G_e = (V_e, A_e)$

Vejamos alguns exemplos, todos com o grafo G_e desenhado na Figura 1.

- **Com $x = 1$ e $y = 4$:** ganha-se invertendo os arcos $(2, 1)$ e $(4, 3)$. O caminho é 1 2 3 4. O número mínimo de jogadas é 2.

Repare que um jogador distraído poderia construir o caminho 1 0 6 5 4, invertendo os arcos $(0, 1)$, $(6, 0)$, $(5, 6)$ e $(4, 5)$. Mas perderia o jogo porque teria invertido 4 arcos (e há uma solução com menos inversões).

- **Com $x = 2$ e $y = 7$:** ganha-se invertendo o arco $(4, 3)$. O caminho é 2 3 4 5 6 7. O número mínimo de jogadas é 1.
- **Com $x = 4$ e $y = 1$:** ganha-se não invertendo arcos. O caminho é 4 5 6 0 1. O número mínimo de jogadas é 0.
- **Com $x = 7$ e $y = 6$:** ganha-se não invertendo arcos. O caminho é 7 6. O número mínimo de jogadas é 0.

Neste problema, pretende-se que implemente uma função, $\text{minReverses}(G, x, y)$, que recebe:

- um grafo G , orientado, não pesado e fracamente conexo, e
- dois vértices distintos, x e y .

A função deve retornar o número mínimo de jogadas para haver um caminho de x para y no último grafo obtido. **O corpo da função *minReverses* deve chamar:**

- uma ou várias funções que constroem grafos;
- um ou vários algoritmos de grafos estudados, como se eles estivessem numa biblioteca (mesmo que esses algoritmos retornem resultados que não interessam para resolver este problema e sejam menos eficientes do que poderiam ser para este caso).

- (a) [1 valor] **Desenhe o(s) grafo(s)** que seria(m) construído(s) durante a execução de *minReverses*(G_e , 1, 4), onde G_e é o grafo do exemplo. Desenhar um grafo é representá-lo da forma habitual (como na Figura 1).

- (b) [2.5 valores] Implemente a função *minReverses* (em pseudo-código). Chame a ou as várias funções que constroem grafos, mas não a(s) implemente (porque já ilustrou o(s) grafo(s) criado(s) na alínea anterior). Não implemente nenhum algoritmo de grafos; recorra aos algoritmos estudados, chamando as respetivas funções, sem as alterar.

```
int minReverses( Digraph graph, Node x, Node y ) {
```

Pergunta 6 O Sr. Yan, dono de um grande restaurante chinês, quer passar a dar três pauzinhos a cada cliente: para além do par habitual, constituído por dois pauzinhos do mesmo tamanho, haverá um terceiro pauzinho mais comprido. O problema é que, para usar apenas os pauzinhos existentes no restaurante, as regras sobre os tamanhos relativos dos pauzinhos têm de ser relaxadas.

Cada cliente receberá um conjunto C com exatamente três pauzinhos, para seu uso exclusivo. Se os comprimentos desses pauzinhos forem:

$$x, y \text{ e } z \quad (\text{com } x \leq y \leq z),$$

a *imperfeição do conjunto* C é $(y - x)^2$, porque os pauzinhos mais curtos deveriam ser do mesmo tamanho e o pauzinho extra será sempre o mais comprido (ou um dos mais compridos).

Nestes tempos de pandemia, em que o número máximo k de clientes é rígido, o Sr. Yan quer formar k conjuntos de pauzinhos, $C_1, C_2, \dots, C_k$, tais que a soma das imperfeições desses conjuntos seja a menor possível. Se chamarmos *desequilíbrio* à soma das imperfeições dos conjuntos, o Sr. Yan quer k conjuntos de pauzinhos (disjuntos dois a dois e cada um com três pauzinhos) que minimizem o *desequilíbrio*.

Suponha que o número máximo de clientes é 2 e que há 8 pauzinhos no restaurante, denotados pelos números $1, 2, \dots, 8$ e com os comprimentos indicados na tabela seguinte.

Pauzinho	1	2	3	4	5	6	7	8
Comprimento	14	20	35	38	38	39	39	43

Há muitas maneiras de formar 2 conjuntos (disjuntos) de 3 pauzinhos cada, mas nenhuma permite obter um *desequilíbrio* inferior a 9. Veja os seguintes exemplos (recordando que a *imperfeição* de um conjunto é sempre calculada com os dois pauzinhos mais curtos):

Conjuntos de Pauzinhos	Desequilíbrio dos Conjuntos
$\{1, 2, 6\}$ e $\{4, 5, 8\}$	$(20 - 14)^2 + (38 - 38)^2 = 36$
$\{3, 4, 8\}$ e $\{5, 6, 7\}$	$(38 - 35)^2 + (39 - 38)^2 = 10$
$\{3, 4, 5\}$ e $\{6, 7, 8\}$	$(38 - 35)^2 + (39 - 39)^2 = 9$

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

- no número máximo k de clientes (que é um inteiro positivo) e
- numa sequência de números inteiros positivos, que correspondem aos comprimentos dos pauzinhos existentes no restaurante,

$$S = (c_1 \ c_2 \ \cdots \ c_n) \quad (\text{com } n \geq 3k),$$

calcule o menor *desequilíbrio* possível de k conjuntos de pauzinhos (disjuntos dois a dois e cada um com três pauzinhos). Pode assumir que a sequência S está ordenada, por ordem crescente ou decrescente (podendo haver elementos iguais).

Para o exemplo dado, em que $k = 2$, $S = (14 \ 20 \ 35 \ 38 \ 38 \ 39 \ 39 \ 43)$ e $n = 8$, o valor da função pretendida é 9.

- (a) [2.7 valores] Defina a função pretendida e indique claramente o que representa cada uma das variáveis que utilizar. Se assumir que a sequência S está ordenada, indique se a ordenação é crescente ou decrescente.

- (b) [0.3 valores] Indique a chamada inicial (a chamada que resolve o problema).