Desenho e Análise de Algoritmos 2020/2021 – Janeiro de 2021

Exame Modelo Duração: 2h

Número mecanográfico: | | | | | | | | | |

Nome completo: _____

Grupo 1 - Análise Assintótica *(10%)*

1.1. Para os pares de funções seguintes, escreva na caixa em branco **Θ**, $\mathcal{O}$ ou **Ω** para tornar a afirmação verdadeira. Note que se as funções em causa tiverem uma relação Θ, não deverá escrever $\mathcal{O}$ ou Ω.

$n^4 = \square\ (4^n)$ $\quad 2^n = \square\ (2^{n+2})$ $\quad 42 = \square\ 24$ $\quad \log(\log n^2) = \square\ (\log n^2)$ $\quad \sqrt{n} = \square\ (\log_4 n)$

1.2. Previsão de Tempo de Execução.
Imagine que tem um programa que demora **1 segundo** para um dado input. Se o **tamanho do input triplicar**, quanto tempo estima que o programa demore para cada uma das seguintes complexidades?
(pode indicar número aproximado ou uma expressão com variáveis)

Se tiver complexidade **linear**, deverá demorar cerca de ____ segundos.

Se tiver complexidade **linearítmica**, deverá demorar cerca de ____ segundos.

Se tiver complexidade **quadrática**, deverá demorar cerca de ____ segundos.

Se tiver complexidade **cúbica**, deverá demorar cerca de ____ segundos.

1.3. Dividir para Conquistar.
Considere a seguinte função recursiva cujo objectivo é somar todos os elementos de um array $v[]$ com n números inteiros:

```
somaArray(v):
    Se tamanho(v) = 1 então retorna ____
    m1 ← somaArray(metade esquerda de v)
    m2 ← somaArray(metade direita de v)
    retorna ____
```

a) Complete as duas caixas de texto em branco a seguir às instruções `retorna` de modo a tornar o algoritmo correcto para a tarefa de descobrir a soma de todos os elementos do array.

b) Escreva uma recorrência $T(n)$ que descreva o tempo de execução da função recursiva `somaArray(v)`. Pode assumir que a divisão do array em duas metades demora tempo constante. ____

c) Indique a complexidade temporal da sua função `somaArray(v)`: ____

Grupo 2 - Ordenação *(10%)*

2.1. Explique as ideias chave do algoritmo **Counting Sort** e indique a sua complexidade temporal.

2.2. Pesquisa Binária. Tem um array `v[]` ordenado de forma **decrescente** e que pretende descobrir se um elemento `x` existe no array. **Escreva o código** necessário para fazer uma pesquisa binária, devolvendo a posição do número no array, ou -1 caso não o encontre. Pode usar pseudo-código, C, C++ ou Java. As posições do array começam em zero.

2.3. Método da Bisseção. Complete os **espaços em branco**.

O método da bisseção usa uma ideia semelhante à pesquisa binária para encontrar _____. Se o intervalo inicial for $[a, b]$ ($f(a)$ e $f(b)$ de sinais opostos), vai considerar os intervalos _____ e _____, continuando para o primeiro destes intervalos quando _____ ou para o segundo caso contrário. Ao fim de n iterações, o intervalo válido que o algoritmo considera ficou reduzido de um tamanho inicial de $b - a$ para um tamanho de _____.

(pode usar o resto desta página para a continuação de respostas dos grupos 1 e 2 que necessitem de mais espaço)

Num. Mec. | | | | | | | | | | Nome:

Grupo 3 - Algoritmos Greedy *(10%)*

3.1. Problema do Troco.

Considere que tem disponíveis uma quantidade "infinita" de moedas de um conjunto de quantias S e que pretende fazer uma quantia K com essas mesmas moedas. **Um algoritmo greedy possível é em cada passo escolher a maior moeda que não faz passar da quantia possível.**

a) Seja $S = \{1, 2, 5, 20, 50, 100, 200\}$ (o conjunto de moedas do euro). O algoritmo greedy iria usar **quantas moedas** para fazer a quantia **143**?

b) O algoritmo greedy nem sempre dá a resposta ótima, ou seja, uma que minimize o número de moedas necessárias. Indique um **contra-exemplo**, ou seja, um conjunto de moedas S e uma quantia k onde o algoritmo falhe:

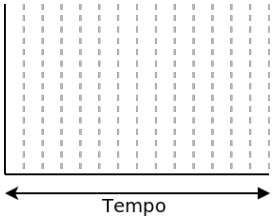
$S =$ $K =$ Nº Moedas Greedy: Nº Moedas Ótimo:

3.2. Interval Scheduling (planeamento de intervalos)

Considere o problema de planeamento de intervalos, onde tem um conjunto de n atividades, cada uma começando no tempo s_i e terminando no tempo f_i. Pretende descobrir **o maior subconjunto de atividades sem sobreposições**.

a) Para cada uma das seguintes **estratégias greedy erradas**, indique um **contra-exemplo** (um caso onde a solução obtida não seja ótima). Desenhe segmentos representando as atividades no tempo, usando letras para as identificar.

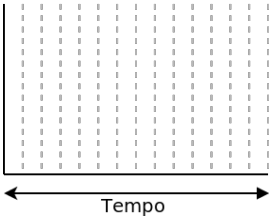
[Início mais cedo]
Alocar por ordem crescente de s_i



Dá como output o seguinte conjunto:

Um conjunto (válido) melhor seria:

[Intervalo mais pequeno]
Alocar por ordem crescente de $f_i - s_i$



Dá como output o seguinte conjunto:

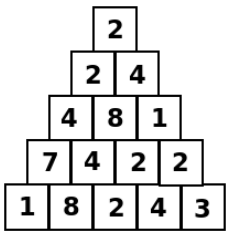
Um conjunto (válido) melhor seria:

b) Indique uma estratégia greedy que dê sempre **resposta ótima**:

Grupo 4 - Programação Dinâmica *(10%)*

4.1. Pirâmide de Números.

Considere a pirâmide de números da figura. Queremos ir do topo até à base sendo que de uma posição apenas se pode ir para uma das duas posições adjacentes imediatamente abaixo.



a) Quantos **caminhos diferentes** existem entre o topo e a base? Justifique a sua resposta.

b) Imagine que quer descobrir o nº caminhos diferentes entre topo e base **passando apenas por números pares**.

Para resolver este problemas pode usar **programação dinâmica** e uma **matriz auxiliar**. Indique uma definição recursiva, como preencheria a matriz para este exemplo, onde ficaria a resposta e qual o número de caminhos que descobriu.

Matriz auxiliar m[5][5]

Definição recursiva da solução: $m(x, y) =$

Onde fica a resposta final na matriz m?

Nº caminhos passando só por números pares:

Grupo 5 - Árvores Binárias de Pesquisa Equilibradas *(10%)*

5.1. Explique as ideias chave de uma **árvore AVL**, indicando que restrições impõe, como mantém essas propriedades e quais as suas complexidades temporais para uma inserção ou remoção.

5.2. Indique **uma vantagem e uma desvantagem** das árvores AVL em relação às árvores Red-Black.

(pode usar o resto desta página para a continuação de respostas dos grupos 3, 4 e 5 que necessitem de mais espaço)

Grupo 6 - Conceitos de Grafos *(5%)*

6.1. Explique o que é o **diâmetro** de um grafo e desenhe um grafo com diâmetro 3 onde todos os nós têm grau > 1.

6.2. Representação de Grafos

Complete cada uma das caixas em branco com as palavras **LISTA** ou **MATRIZ** para indicar se a resposta deve ser uma lista de adjacências ou uma matriz de adjacências.

[ ] ocupa menos memória

[ ] é melhor para verificação se existe ligação entre um par de nós

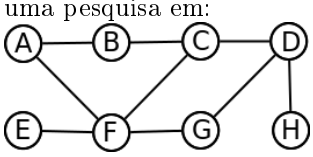
[ ] é melhor para remover uma única aresta

[ ] é melhor para apagar todas as arestas ligadas a um dado nó

Grupo 7 - Pesquisas em Profundidade e em Largura *(15%)*

7.1. Conceitos Básicos

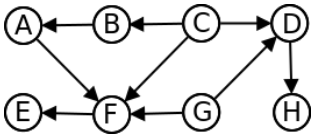
a) Considere o grafo da figura seguinte. Imagine que **começa uma pesquisa no vértice** (G) e que os nós vizinhos são sempre **percorridos por ordem alfabética**. Indique a **ordem** em que os nós seriam visitados se fosse usada uma pesquisa em:



Profundidade: [ ][ ][ ][ ][ ][ ][ ][ ]

Largura: [ ][ ][ ][ ][ ][ ][ ][ ]

7.2. Ordenação Topológica. Indique duas possíveis ordenações topológicas para o grafo seguinte:

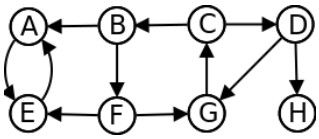


Duas Ordenações topológicas:

1) [ ][ ][ ][ ][ ][ ][ ][ ]

2) [ ][ ][ ][ ][ ][ ][ ][ ]

7.3. Componentes Fortemente Conexos. Indique os componentes fortemente conexos (CFCs) do grafo seguinte:



Nº CFCs: [ ]

Nós de cada CFC: [ ]

7.4. Código de Pesquisa em Profundidade

Escreva (em código ou pseudo-código) uma função $dfs(G, v)$ para fazer uma pesquisa em profundidade a partir do nó v de um grafo G. **A função deve devolver o número de nós que foram visitados.**

7.5. Código de Pesquisa em Largura

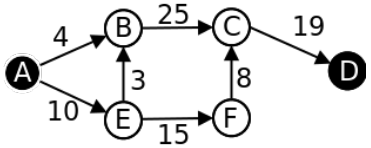
Escreva (em código ou pseudo-código) uma função $bfs(G, v, u)$ para fazer uma pesquisa em largura a partir do nó v de um grafo não pesado G. **A função deve devolver a distância de v a u.**

(pode usar o resto desta página para a continuação de respostas dos grupos 6 e 7 que necessitem de mais espaço)

Num. Mec. | | | | | | | | | | Nome:

Grupo 8 - Distâncias Mínimas *(10%)*

8.1. Algoritmo de Dijkstra. Considere o grafo da figura seguinte.



a) Aplique manualmente o algoritmo de **Dijkstra** supondo que começa a partir do nó (A). A primeira linha da tabela indica com que valores deve iniciar a estimativa de distância de cada nó ao nó inicial.

Iteração	Nó Escolhido	Nó Predecessor *(pai)*	Estimativa de distância ao nó A (depois de relaxadas as arestas do nó escolhido)					
			A	B	C	D	E	F
Inicialização	———	———	0	∞	∞	∞	∞	∞
1ª	A	———						
2ª								
3ª								
4ª								
5ª								

b) Qual é a distância mínima que o algoritmo de Dijkstra calculou de (A) para (B)? ☐

Qual é o caminho que dá origem a essa distância? ☐

c) Qual a complexidade temporal do Dijkstra se usarmos heaps? ☐

8.2. Algoritmo de Bellman-Ford.

Considere o seguinte pseudo-código incompleto para usar o algoritmo de **Bellman-Ford** no grafo G a partir do nó s.

Bellman-Ford(G, s):
  Para todos os nós v de G **fazer**: $v.dist \leftarrow \infty$
  $s.dist \leftarrow 0$
  Para $i \leftarrow 1$ **até** ☐ **fazer**:
    Para todas as arestas (u, v) de G **fazer**:

☐

a) Complete os espaços em branco de cima com código ou pseudo-código.

b) Justifique o valor que colocou na primeira caixa (limite do ciclo de i), explicando porque não pode ser um número menor, e porque é que um número maior seria escusado.)

☐

Grupo 9 - Árvores de Suporte de Custo Mínimo (MST) *(5%)*

c) O **algoritmo de Prim** adiciona um nó de cada vez para formar uma MST. Supondo que começava no nó (A), indique qual a sucessão de nós que vai sendo adicionada pelo algoritmo até formar uma MST no grafo da figura.

1º nó adicionado: ☐ através da aresta ☐

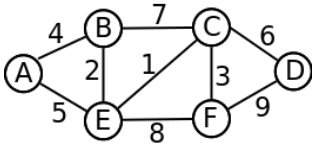
2º nó adicionado: ☐ através da aresta ☐

3º nó adicionado: ☐ através da aresta ☐

4º nó adicionado: ☐ através da aresta ☐

5º nó adicionado: ☐ através da aresta ☐

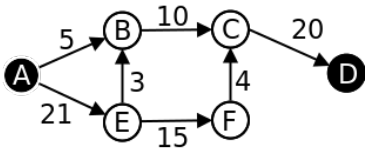
d) Qual é o **custo da árvore mínima de suporte** no grafo dado? ☐



Grupo 10 - Fluxo Máximo *(10%)*

10.1. Método de Ford-Fulkerson.

a) Imagine que quer descobrir o **fluxo máximo** entre A e D no grafo do lado. Aplique o método de **Ford-Fulkerson** por 2 iterações, indicando em cada passo o que fez. No **grafo residual** deve desenhar as arestas e os respectivos pesos depois de aplicado o caminho de aumento que indicou.

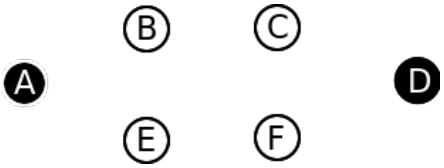


1ª Iteração

Caminho de aumento:

Valor do fluxo aplicado no caminho indicado:

Grafo residual:

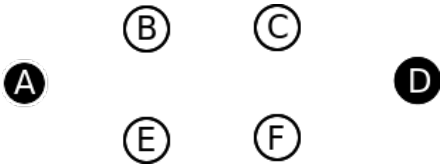


2ª Iteração

Caminho de aumento:

Valor do fluxo aplicado no caminho indicado:

Grafo residual:



b) No final das 2 iterações, qual o valor do **fluxo total** entre A e D?

c) O fluxo que indicou é o **fluxo máximo**? Justifique.

(pode usar o resto desta página para a continuação de respostas dos grupos 8, 9 e 10 que necessitem de mais espaço)

Grupo 11 - Perguntas de Valorização *(5%)*

11.1. Esboce uma **prova da correção** da escolha greedy que indicou na pergunta 3.2.b (repita o essencial da escolha aqui: esta folha será corrigida separadamente).

11.2. Imagine que quer calcular o **maior número de caminhos entre** u **e** v **num grafo** G **que não repitam nenhum nó** (ao invés de simplesmente não repetir arestas). Seria possível usar um algoritmo de fluxo máximo? Justifique, indicando como proceder.

(pode usar o resto desta página para a continuação de respostas do grupo 11 que necessitem de mais espaço)

(pode usar esta página para a continuação de respostas do grupo 11 que necessitem de mais espaço)