Editado em 9 de março de 2026

UNIVERSIDADE FEDERAL DE ALAGOAS
INSTITUTO DE COMPUTAÇÃO

Projeto e Análise de Algoritmos

Lista de Exercícios

Prof. Rian Gabriel Pinheiro

2026.1

Instruções

(a) As listas podem ser resolvidas em dupla, neste caso deve-se entregar apenas um trabalho!

(b) Mencione os teoremas e propriedade usadas para justificar usas afirmações.

(c) Algoritmos devem ser escritos em pseudocódigo bem comentado, ou informalmente, mas com precisão.

(d) Cada entrega corresponde a 2 seções (20 questões) do trabalho.

(e) Qualquer modificação das duplas deve ser informada.

(f) As resoluções devem ser entregues escritas à mão (com exceção questões que pedem explicitamente uma implementação ou gráficos) nos dias de prova.

(g) Qualquer tentativa de fraude implicará em nota *zero* na parte correspondente.

(h) Cada questão correta irá incrementar 0,10 ponto na respectiva prova.

Parte I.
1ª VA

1. Corretude de Algoritmos

1.1 . Faça uma pesquisa (escreva pelo menos 10 linhas) sobre al-Khorezmi (também al-Khwarizmi), o homem de cujo nome deriva a palavra "algoritmo". Mostre o que as origens das palavras "algoritmo" e "álgebra" têm em comum.

1.2 . Prove que:

- $n^3 + 2n$ é divisível por 3 para todo $n \geq 0$, por indução.
- Se $2 \mid 3m$ (2 divide $3m$) então $2 \mid m$, por contraposição.
- A soma de 3 números consecutivos é múltiplo de 3, use prova direta.

1.3 . Considere o seguinte algoritmo:

```
1: procedure ALGORITMO X(vetor A[1,...,n], inicio, fim)
2:     if inicio = last then
3:         return A[inicio]
4:     end if
5:     meio ← inicio + (fim − incio)/2
6:     a ← X(A, inicio, meio)
7:     b ← X(A, meio +1, fim)
8:     if a < b then
9:         return b
10:    else
11:        return a
12:    end if
13: end procedure
```

Explique o que ele faz e prove sua corretude.

1.4 . Prove a corretude do algoritmo de Horner para a avaliação de polinômios. $P(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$.

```
1: procedure ALGORITMO DE HORNER(vetor A[0,...,n], real x)
2:     p ← A[n]
3:     for i ← n − 1 → 0 do
4:         p ← p ∗ x + A[i]
5:     end for
6:     return p
7: end procedure
```

1.5 . Escreva uma função recursiva que imprima uma régua de ordem n no intervalo $[0..2^n]$ e prove sua corretude. O "traço" no ponto médio da régua deve ter comprimento n, os traços nos pontos médios dos subintervalos superior e inferior devem ter comprimento $n-1$, e assim por diante. Abaixo um exemplo de régua de ordem 4.

```
0       .
1       . -
2       . --
3       . -
4       . ---
5       . -
6       . --
7       . -
8       . ----
9       . -
10      . --
11      . -
12      . ---
13      . -
14      . --
15      . -
16      .
```

1.6 . Prove a corretude do algoritmo Conversor Decimal-Binário.

```
1: procedure CONVERSOR D-B(inteiro n)
2:     t ← n
3:     k ← 0
4:     zere todos os bits de b
5:     while t > 0 do
6:         k ← k + 1
7:         b[k] ← t mod 2
8:         t ← t ÷ 2
9:     end while
10:    return b
11: end procedure
```

1.7 . A sequencia de Fibonacci é definida da seguinte forma: $f_0 = 0$; $f_1 = 1$ e $f_{i+2} = f_{i+1} + f_i$, $\forall i \geq 0$. Prove que para todo $n \geq 1$ temos:

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n = \begin{pmatrix} f_{n+1} & f_n \\ f_n & f_{n-1} \end{pmatrix},$$

em que o lado esquerdo representa a n-ésima potência de uma matriz 2×2.

1.8 . Para cada $n \geq 1$, considere um tabuleiro (de xadrez) quadrado de tamanho $2^n \times 2^n$ em que uma única casa está faltando. Prove que o tabuleiro pode ser preenchido com peças em formato de "L". Cada peça ocupa três casas do tabuleiro.

1.9 . O BIN PACKING é um problema cuja entrada consiste em: n itens com tamanhos $s_1, s_2, \ldots, s_n$ em que $s_i \in [0,1]$. O objetivo é encontrar o menor número de "*bins*" (caixas) unitárias para armazenar os n itens. Dado os algoritmos a seguir, mostre que eles não encontram a solução ótima do problema, ou seja, encontre contraexemplos para cada um dos seguinte algoritmos para o problema.

- Coloque na *bins* os elementos em ordem da esquerda para a direita, se ele couber, caso contrário tente na próxima *bin*;
- Coloque na *bin* mais livre o maior elemento;
- Coloque o menor elemento na **bin** mais livre.

1.10 . Considere o algoritmo de Ulam, ele termina? De fato, conjectura-se que seguindo o algoritmo, sempre será obtida a sequencia $4, 2, 1$ (Conjectura de Collatz). Ex: Para o valor $a = 22$, será obtida a seguinte sequencia: $22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1$. Como a prova do término do algoritmo consiste em um difícil problema matemático em aberto. **Implemente** um **teste exaustivo** mostrando que para qualquer `unsigned shot int` (1 a 65535) de entrada o algoritmo para. Escreva um pequeno relato informando

o tamanho da maior sequência encontrada, o valor na qual a maior sequência foi obtida, a média dos tamanhos das sequências e o tempo de execução.

```
1: procedure ALGORITMO DE ULAM(inteiro positivo a)
2:     x ← a
3:     while Os três últimos valores de x não for 4,2,1 do
4:         if x for par then
5:             x ← x/2
6:         else
7:             x ← 3x + 1
8:         end if
9:     end while
10: end procedure
```

2. Complexidade de Algoritmos

2.1 . Em cada caso, indique se $f(n) = O(g(n))$ ou $f(n) = \Omega(g(n))$, ou ambos (neste caso $f(n) = \Theta(g(n))$):

	$f(n)$	$g(n)$
(1)	$n-10$	$n-200$
(2)	$n^{1/2}$	$n^{2/3}$
(3)	$100n+\log n$	$n+(\log n)^2$
(4)	$\log 2n$	$\log 3n$
(5)	$10\log n$	$\log(n^2)$
(6)	$n^{1,01}$	$n\log^2 n$
(7)	$n^2/\log n$	$n(\log n)^2$
(8)	$n^{0,1}$	$(\log n)^{10}$
(9)	$(\log n)^{\log n}$	$n/\log n$
(10)	$\sqrt{n}$	$(\log n)^3$
(11)	$n^{1/2}$	$5^{\log_2 n}$
(12)	$n2^n$	3^n
(13)	2^n	2^{n+1}
(14)	$n!$	2^n
(15)	$(\log n)^{\log n}$	$2^{(\log_2 n)^2}$
(16)	$\sum_{i=1}^{n} i^k$	n^{k+1}

2.2 . Escreva o pseudocódigo do *algoritmo de busca binária* iterativo ou recursivo e demostre que o pior caso da busca binária é $\Theta(\lg n)$.

2.3 . Descreva um algoritmo de tempo $\Theta(n \lg n)$ que, dado um conjunto S de n inteiros e um outro inteiro x, determine se existe ou não dois elementos em S cuja soma seja exatamente x.

2.4 . Seja $A[1..n]$ um arranjo de n números distintos. Se $i < j$ e $A[i] > A[j]$, então o par (i, j) é denominado *inversão* de A.

a) Dado o conjunto com os elementos $\{1, 2, \ldots, n\}$, qual arranjo tem o maior números de inversões? Quantas inversões ele tem?

b) Dê um algoritmo que determine o número de inversões em qualquer permutação com os n elementos em tempo do pior caso $\Theta(n \lg n)$.

2.5 . Considere o seguinte trecho de código:

```
1: procedure PROG1(inteiro positivo k)
2:     if k == 1 then
3:         Print("Oi") return
4:     else
5:         for i ← 1 → k do
6:             Print("Oi")
7:         end for
8:         Prog1(k − 1)
9:     end if
10: end procedure
```

Seja $T(n)$ o número de vezes que a palavra OI é impressa quando `Prog1` é chamado com parâmetro n. Calcule $T(1)$ e determine uma relação entre $T(n)$ e $T(n-1)$ para $n > 1$. Com base na relação encontrada, utilize indução para mostrar que $T(n) \leq n^2$, para todo n maior ou igual a 1.

2.6 . Considere o seguinte trecho de código:

```
1: procedure CONT(inteiro positivo n)
2:     c ← 0
3:     for l ∈ [1, n] do
4:         for i ∈ [1, n − l] do
5:             for k ∈ [1, i + l] do
6:                 c ← c + 1
7:             end for
8:         end for
9:     end for
10:    return c
11: end procedure
```

Dado um valor n, qual será o valor da variável c? Mostre as contas.

2.7 . Considere o seguinte algoritmo , cujo argumento n é um inteiro positivo.

```
1: procedure ALG(n)
2:     c ← 0
3:     for i ← 1 → ⌊log n⌋ do
4:         for j ← i → i + 5 do
5:             for k ← 1 → i^2 do
6:                 c ← c + 1
7:             end for
8:         end for
9:     end for
10: end procedure
```

Para um dado valor de n, quantos asteriscos serão impressos em uma chamada de ASTERISCO(n)?

2.8 . Considere o seguinte algoritmo recursivo, cujo argumento n é um inteiro positivo.

```
1: procedure ASTERISCO(n)
2:     if n > 0 then
3:         ASTERISCO(n − 1)
4:         for i ← 1 → n do
5:             imprima "*"
6:         end for
7:         ASTERISCO(n − 1)
8:     end if
9: end procedure
```

Para um dado valor de n, quantos asteriscos serão impressos em uma chamada de ASTERISCO(n)?

2.9 . Sejam $f(n)$ e $g(n)$ funções assintoticamente não negativas. Usando a definição básica da notação Θ, prove que $\max(f(n), g(n)) = \Theta(f(n) + g(n))$.

2.10 . Mostre que

a) A solução de $T(n) = T(n-1) + n$; $T(1) = 1$ é $O(n^2)$.

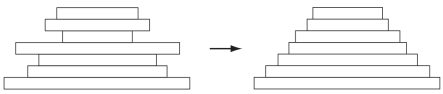
b) A solução de $T(n) = T(\lceil n/2 \rceil) + 1$; $T(1) = 1$ é $O(\lg n)$.

c) A solução de $2T(\lfloor n/2 \rfloor) + n$; $T(1) = 1$ é $O(n \lg n)$.

3. Divisão e Conquista

3.1 . Explique a relação da técnica Divisão-e-Conquista com paralelismo. Como utilizar o paralelismo em um algoritmo divisão-e-conquista? Por que elas funcionam bem em conjunto?

3.2 . Existem n panquecas, todas de tamanhos diferentes, empilhadas umas sobre as outras. Você pode colocar uma espátula sob uma das panquecas e virar a pilha inteira acima da espátula. O objetivo é arranjar as panquecas de acordo com o tamanho, com a maior na parte inferior. A Figura mostra uma instância do quebra-cabeça para $n = 7$. Projete um algoritmo para resolver este problema e determine o número de operações feitas pelo algoritmo no pior caso.



3.3 . Suponha que você tenha os resultados de um torneio concluído no qual n equipes jogaram entre si uma vez. Assumindo que não houve empate, mostre um algoritmo que liste os times em uma sequência de forma que todos ganhem o jogo com o time listado imediatamente após?

3.4 . Escreva um algoritmo de divisão-e-conquista $O(\log n)$ para computar a^n em que n é um inteiro positivo.

3.5 . São dadas duas listas **ordenadas** de tamanho m e n. Dê um algoritmo de tempo $O(\log m + \log n)$ para computar o k-ésimo menor elemento da união das duas listas.

3.6 . Suponha que esteja escolhendo entre os seguintes três algoritmos:

- Algoritmo A resolve problemas dividindo-os em cinco subproblemas de metade do tamanho, solucionando cada subproblema recursivamente e, então, combinando as soluções em tempo linear.
- Algoritmo B resolve problemas de tamanho n resolvendo recursivamente dois subproblemas de tamanho $n-1$ e, então, combinando as soluções em tempo constante.
- Algoritmo C soluciona problemas de tamanho n dividindo-os em nove subproblemas de tamanho $n/3$, resolvendo recursivamente cada subproblema e, então, combinando as respostas em tempo $O(n^2)$.

Qual o tempo de execução de cada um desses algoritmos (em notação O) e qual você escolheria?

3.7 . Dado $A[1, \ldots, n]$, um vetor ordenado de inteiros distintos, você quer saber se existe um índice i para o qual $A[i] = i$. Dê um algoritmo de divisão-e-conquista que execute em tempo $O(\log n)$.

3.8 . Mostre que qualquer vetor de inteiros $x[1 \ldots n]$ pode ser ordenado em tempo $O(n + M)$, onde

$$M = \max_i x_i - \min_i x_i.$$

3.9 . Você está consultando para uma pequena empresa de investimentos. Eles estão fazendo uma simulação em que eles olham para n dias consecutivos de uma determinada ação, em algum momento no passado. Cara cada dia $i = 1, 2, \ldots, n$; eles têm o preço p_i da ação neste dia. Suponha que durante este período de tempo, eles queriam comprar 1.000 ações em alguns dias e vender todas essas ações em algum dia (mais tarde). Eles querem saber: Quando eles deveria ter comprado e quando eles deveriam ter vendido, a fim de maximizar os lucros? Por exemplo, suponha que $n = 3, p_1 = 9, p_2 = 1, p_3 = 5$. Veja que deveria retornar "comprar em 2, vender em 3"(compra no dia 2 e vender no dia 3 significa que eles teria feito \$4 por ação, o máximo possível para esse período). Claramente, há um algoritmo simples que leva tempo $O(n^2)$: tentar todos os possíveis pares compra/venda e ver qual deles faz mais dinheiro. Elabore um algoritmo divisão-e-conquista para encontrar os dias de compra e venda com tempo $O(n \log n)$.

3.10 . Mostre como remover todos os elementos duplicados de um vetor em $O(n \log n)$. Você pode modificar um algoritmo de ordenação, mas não pode ordenar e depois remover.

4. Algoritmos Gulosos

4.1 . As sentenças seguintes podem ou não estar corretas. Em cada caso, prove a sentença (se ela for correta) ou forneça um contraexemplo (se não for correta). Sempre considere o grafo $G = (V, E)$ não-direcionado e conexo. Não considere que os pesos nas arestas sejam distintos, a menos que isso seja explicitamente afirmado.

a) Se um grafo G tem mais do que $|V| - 1$ arestas e existe uma única aresta de maior peso, então esta aresta não pode ser parte da árvore geradora mínima.

b) Se G tem um ciclo com uma aresta de maior peso única e, então e não pode ser parte de nenhuma AGM.

c) Seja e qualquer aresta de peso mínimo em G. Então e tem de ser parte de alguma AGM.

d) Se a aresta de menor peso em um grafo é única, então ela tem de ser parte da todas as AGMs.

e) Se e é parte de alguma AGM de G, então ela tem de ser a aresta de menor peso através de algum corte de G.

f) Se G tem um ciclo com uma aresta mais leve única e, então e tem de ser parte de todas as AGM.

g) A árvore de caminhos mínimos computada pelo algoritmo de Dijkstra é necessariamente uma AGM.

h) O caminho mínimo entre dois nós é necessariamente parte de alguma AGM.

i) O algoritmo de Prim funciona corretamente quando existem arestas negativas.

j) (Para qualquer $r > 0$, defina um r-caminho cujas arestas tenham todas peso $< r$.) Se G contém um r-caminho do nó s até t, então toda AGM de G tem de conter também um r-caminho do nó s até o nó t.

4.2 . Considere um grafo não-direcionado $G = (V, E)$ com pesos de aresta não-negativos $w_e \geq 0$. Suponha que você computou uma árvore geradora mínima de G e que também computou os caminhos mínimos para todos os nós partindo de um particular nó $s \in V$. Agora suponha que cada peso de aresta seja aumentado em uma unidade: os novos pesos são $w'_e = w_e + 1$.

a) Será que a árvore geradora mínima muda? Dê um exemplo para o qual ela muda ou prove que ela não pode mudar.

b) Será que os caminhos mínimos mudam? Dê um exemplo para o qual eles mudam ou prove que isso não pode ocorrer.

4.3 . Seja G uma grafo não-direcionado. Prove que se todas as arestas são distintas, então ele tem uma única árvore geradora mínima.

4.4 . Considere o problema de agendamento de intervalos. Nós temos um conjunto de atividades $\{1, 2, \ldots, n\}$; cada atividade i possui um intervalo de tempo a partir de s_i e termina em f_i. Um agendamento — conjunto de atividades — é dito compatível, se nenhuma atividade se sobrepõem no tempo. O objetivo é determinar um agendamento compatível com o maior número possível de atividades. Projete uma algoritmo para este problema.

4.5 . Este é um problema que ocorre em análise automática de programas. Para um conjunto de variáveis $x_1, \ldots, x_n$, são dadas algumas restrições de igualdade, da forma "$x_i = x_j$" e algumas restrições de desigualdade, da forma "$x_i \neq x_j$". Será que é possível satisfazer todas elas? Por exemplo, as restrições

$$x_1 = x_2,\ x_2 = x_3,\ x_3 = x_4,\ x_1 \neq x_4,$$

não podem ser satisfeitas. Forneça um algoritmo eficiente que tome como entrada m restrições sobre n variáveis e decida se as restrições podem ser satisfeitas.

4.6 . Alice quer dar uma festa e está decidindo quem chamar. Ela tem n pessoas as quais escolher, e ela fez uma lista de quais pares dessas pessoas conhecem uma a outra. Ela quer selecionar o maior número de pessoas possível, sujeito a duas restrições: na festa, cada pessoa deve ter pelo menos outras cinco pessoas que ela conhece e outras cinco pessoas que ela não conhece. Forneça um algoritmo eficiente que tome como entrada a lista das n pessoas e a lista de pares de quem conhece quem e calcule a melhor escolha de convidados para a festa. Dê o tempo de execução em termos de n.

4.7 . Um servidor tem n usuários esperando para serem servidos. O tempo de serviço requerido por usuário é conhecido previamente: é t_i minutos para o usuário i. Portanto se, por exemplo, os usuários são servidos em ordem crescente de i, então o i-ésimo usuário tem de esperar a $\sum_{j=1}^{i} t_j$ minutos. Queremos minimizar o tempo total de espera

$$T = \sum_{n=1}^{n} (\text{tempo gasto pelo usuário } i \text{ na espera}).$$

Forneça um algoritmo eficiente para computar a ordem ótima na qual processar os usuários.

4.8 . Um conjunto *feedback de arestas* de um grafo não-direcionado $G = (V, E)$ é um subconjunto de arestas $E' \subseteq E$ que intercepta todos os ciclos do grafo. Assim, remover as arestas E' tornará o grafo acíclico. Forneça um algoritmo eficiente para o seguinte problema:

Entrada: Grafo não-direcionado $G = (V, E)$ com pesos de aresta positivos w_e.
Saída: Um conjunto feedback de arestas definida $E' \subseteq E$ de peso total mínimo $\sum_{e \in E'} w_e$.

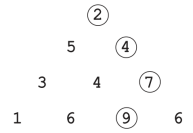
4.9 . Forneça um algoritmo de tempo linear que tome como entrada uma árvore e determine se ela tem um *emparelhamento perfeito*: um conjunto de arestas que tocam cada vértice exatamente uma vez.

4.10 . Prove que o problema da mochila pode ser resolvido em tempo polinomial por um algoritmo guloso, se o peso w_i de cada item i for unitário.

Parte II.
2ª VA

5. Programação Dinâmica

5.1 . Inteiros positivos são arranjados em um triângulo equilátero com n números em sua base, como o mostrado na figura abaixo para $n = 4$. O problema é encontrar a menor soma em uma descida do ápice do triângulo até sua base por meio de uma sequência de números adjacentes (mostrados na figura pelos círculos). Projete um algoritmo de programação dinâmico para este problema.



5.2 . Algumas moedas são espalhadas nas células de um tabuleiro $n \times m$, uma moeda por célula. Um robô, localizado na célula superior esquerda do tabuleiro,

precisa coletar o máximo de moedas possível e trazê-las para a célula inferior direita. Em cada etapa, o robô pode mover uma célula para a direita ou uma célula para baixo de sua localização atual. Quando o robô visita uma célula com uma moeda, ele pega a moeda. Elabore um algoritmo para encontrar o número máximo de moedas que o robô pode coletar e um caminho que ele precisa seguir para fazer isso.

5.3 . Apresente o pseudocódigo do algoritmo de Floyd-Warshall para o problema dos caminhos mínimos entre todos os pares de vértices. Apresente a recorrência para o cálculo da distância.

5.4 . Projete um algoritmo eficiente para encontrar o comprimento do caminho mais longo em um DAG. (Este problema é importante como um protótipo de muitos outros aplicativos de programação dinâmica, pois determina o tempo mínimo necessário para concluir um projeto que compreende tarefas de precedência restrita.)

5.5 . Uma subsequência contígua de uma lista S é uma subsequência feita de elementos consecutivos de S. Por exemplo, se S é

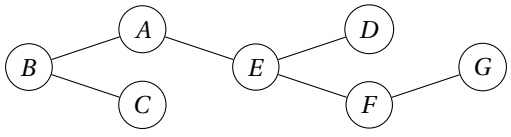
$$5, 15, -30, 10, -5, 40, 10,$$

então $15, -30, 10$ é uma subsequência contígua, mas $5, 15, 40$ não é. Forneça um algoritmo de tempo linear para a seguinte tarefa:
Entrada: Uma lista de números $a_1, a_2, \ldots, a_n$.
Saída: A subsequência contígua de soma máxima (a subsequência de tamanho zero tem soma zero). Para o exemplo anterior, a resposta seria $10, -5, 40, 10$, com uma soma de 55. (Dica: Para cada $j \in \{1, 2, \ldots, n\}$ considere subsequências contíguas terminando exatamente na posição j.)

5.6 . Dadas duas strings $x = x_1 x_2 \cdots x_n$ e $y = y_1 y_2 \cdots y_m$, desejamos encontrar o comprimento da maior substring comum delas, isto é, o maior k para o qual existem índices i e j com $x_i x_{i+1} \cdots x_{i+k-1} = y_j y_{j+1} \cdots y_{j+k-1}$. Mostre como fazer isso em tempo $O(mn)$

5.7 . Uma cobertura de vértices de um grafo $G = (V, E)$ é um subconjunto de vértices $S \subseteq V$ que inclui ao menos uma extremidade de cada aresta de E. Forneça um algoritmo de tempo linear para a seguinte tarefa.

Entrada: Uma árvore não-direcionada $T = (V, E)$.
Saída: O tamanho da menor cobertura de vértices de T.
Por exemplo, na árvore abaixo, as possíveis coberturas de vértice incluem $\{A, B, C, D, E, F, G\}$ e $\{A, C, D, F\}$, mas não $\{C, E, F\}$. A menor cobertura de vértice tem tamanho 3: $\{B, E, G\}$.



5.8 . Forneça um algoritmo de programação dinâmica (note que será pseudo-polinomial) para o problema SUBSET SUM.
Entrada: Um conjunto A com valores inteiros positivos, e um inteiro t.
Questão: Existe um subconjunto $A' \subseteq A$ cuja soma dos valores seja exatamente t?

5.9 . Dado um DAG $G(V, E)$ e dois vértices $s, t \in V$, projete um algoritmo de programação dinâmica que encontre a quantidade de caminhos simples distintos entre s e t.

5.10 . Mostre uma algoritmo de programação dinâmica para o problema do CAMINHO MÁXIMO entre dois vértices s e t (caminho simples). Qual a complexidade do algoritmo (tempo e memória)?

6. Transformação de problemas

6.1 . Maria aposta com João que ela pode fazer o seguinte truque. João recitará $n-1$ números diferentes de 1 a n em uma ordem aleatória e ela será capaz de nomear o único número nesse intervalo que ele terá perdido. Claro, ela terá que realizar a tarefa em sua cabeça, sem fazer anotações. Como ela deve fazer esse truque? Em outras palavras, projete um algoritmo que descubra o número faltante utilizando $O(1)$ de espaço em memória.

6.2 . O Rei Arthur espera n cavaleiros para um jantar anual em Camelot. Infelizmente, alguns dos cavaleiros brigam entre si, e Arthur sabe quem briga com quem. Arthur quer sentar seus convidados ao redor de uma mesa para que dois cavaleiros briguentos não se sentem próximos um do outro. Qual problema pode ser usado para modelar a tarefa do Rei Arthur?

6.3 . Várias famílias saem para jantar juntas. Para aumentar sua interação social, eles gostariam de sentar-se à mesa, de modo que dois membros da mesma família não estivessem na mesma mesa. Mostre como encontrar uma disposição dos assentos que atenda a esse objetivo (ou prove que não existe tal disposição) usando o problema de fluxo máximo. Suponha que o jantar tenha p famílias e que a i-ésima família tenha a_i os membros. Suponha que q mesas são disponíveis e que a meja j-ésima mesa possui capacidade b_j.

6.4 . O problema da coloração em grafos é geralmente declarado como o problema da coloração do vértice: atribua o menor número de cores aos vértices de um dado grafo de forma que não haja dois vértices adjacentes da mesma cor. Considere agora o problema da coloração das arestas: atribua o menor número possível de cores às arestas de um determinado grafo, de modo que duas arestas com o mesmo vértices não tenham a mesma cor. Explique como o problema de coloração de arestas pode ser reduzido a um problema de coloração de vértices.

6.5 . Considere o seguinte problema de programação li-

near.

$$\begin{aligned} \max \quad & 5x+3y \\ \text{s.a} \quad & 5x+2y \geq 0 \\ & x+y \leq 7 \\ & x \leq 5 \\ & x \geq 0 \\ & y \geq 0 \end{aligned}$$

Desenhe a região factível e identifique a solução ótima.

6.6 . A companhia de produtos caninos oferece duas comidas para cachorro: Viralata's e Ração do Sucesso, que são feitas de uma mistura de cereais e carne. Um pacote de Viralata's requer 1 quilo de cereal e 1,5 quilo de carne, e é vendido por $7. Um pacote de Ração do Sucesso usa 2 quilos de cereal e 1 quilo de carne, e é vendido por $6. O cereal bruto custa $1 por quilo e a carne bruta, $2 por quilo. Há também o custo de $1,40 para empacotar o Viralata's e $0,60 para o Ração do Sucesso. Um total de 240.000 quilos de cereal e 180.000 quilos de carne estão disponíveis a cada mês. O único gargalo de produção está no fato de a fábrica poder empacotar apenas 110.000 pacotes de Viralata's por mês. Desnecessário dizer, a gerência gostaria de maximizar o lucro.

a) Formule o problema como um programa linear em duas variáveis.

b) Desenhe a região factível, dê as coordenadas de cada vértice, e circule o vértice que maximiza o lucro. Qual o lucro máximo possível?

6.7 . O problema das 8 DAMAS consistem em colocar 8 damas em um tabuleiro de xadrez. Modele este problema utilizando Programação por Restrição.

6.8 . Leia o artigo da wikipedia sobre o puzzle Kakuro `https://en.wikipedia.org/wiki/Kakuro` e modele exemplo dado utilizando Programação por Restrição.

6.9 . Modele o problema da MOCHILA utilizando programação linear inteira.

6.10 . Modelo o problema da CLIQUE MÁXIMA utilizando programação linear inteira.

7. NP-completude

7.1 . Responda cada um dos itens abaixo e dê uma justificativa para as respostas.

- O que significa dizer que um problema Π pode ser polinomialmente reduzido a um problema Π' ?
- Defina as classes P, NP e problema NP-completo.
- $P \cap NP = \emptyset$?

7.2 . Responda cada um dos itens abaixo e dê uma justificativa para as respostas.

- Se um problema Π pode ser polinomialmente reduzido a um problema Π' e Π' está em P então Π está em P?
- Se um problema Π pode ser polinomialmente reduzido a um problema Π' e Π' é NP-completo então Π é NP-completo?
- Há problemas em NP que não são NP-completos?
- Existem problemas NP-completos em P?

7.3 . Encontre 5 problemas NP-completos não estudados na disciplina (aula e lista). Para cada um deles, descreva-o formalmente (entrada e questão) e apresente uma ilustração de uma instância e uma solução.

7.4 . Considere o seguinte algoritmo força bruta para resolver o problema do número COMPOSTO: Verifique inteiros sucessivos de 2 a $\lfloor n/2 \rfloor$ como possíveis divisores de n. Se um deles divide n , retorna SIM (ou seja, o número é composto); se nenhum deles o fizer, retorne NÃO. Por que esse algoritmo não coloca o problema na classe P?

7.5 . Considere os problemas a seguir. PROBLEMA PARTIÇÃO(A, n):
Dado um vetor $A[1, \ldots, n]$ de números inteiros positivos, decidir se existe um subconjunto I de $\{1, \ldots, n\}$ tal que

$$\sum_{i \in I} A[i] = \sum_{i \notin I} A[i].$$

PROBLEMA TRI-PARTIÇÃO(A, n):
Dado um vetor $A[1, \ldots, n]$ de números inteiros positivos, decidir se existem subconjuntos disjuntos I e J de $\{1, \ldots, n\}$ tais que

$$\sum_{i \in I} A[i] = \sum_{i \in J} A[i] = \sum_{i \notin I \cup J} A[i].$$

Sabe-se que o PROBLEMA PARTIÇÃO é NP-completo. Um aluno alega que o PROBLEMA TRI-PARTIÇÃO também é NP-completo. O aluno está certo? Justifique cuidadosamente a sua resposta.

7.6 . Busca versus decisão. Suponha que você tenha um procedimento que execute em tempo polinomial e que informe se um grafo tem ou não um caminho Hamiltoniano. Mostre que você pode usá-lo para desenvolver um algoritmo de tempo polinomial para o PROBLEMA DO CAMINHO HAMILTONIANO em sua versão de busca (que retorna o caminho propriamente, se ele existe).

7.7 . O PROBLEMA DA ÁRVORE GERADORA COM RESTRIÇÃO DE GRAU é o seguinte. Entrada: Um grafo não-direcionado $G(V, E)$.
Saída: Uma árvore geradora de G na qual cada nó tem grau $\leq k$, se uma árvore desse tipo existe.

Mostre que para todo $k \geq 2$ o problema é NP-difícil.

7.8 . Uma pipa é um grafo sobre um número par de vértices, digamos $2n$, nos quais n dos vértices formam uma clique e os restantes n vértices são conectados em um "rabo" que consiste em um caminho incidente a um dos vértices da clique. Dado um grafo e um objetivo g, o PROBLEMA DA PIPA pede que se encontre um subgrafo que seja uma pipa e que contenha $2g$ nós. Prove que PIPA é NP-completo.

7.9 . No PROBLEMA CONJUNTO INCIDENTE, é dada uma família de conjuntos $\{S_1, S_2, \ldots, S_n\}$ e um inteiro b, e desejamos encontrar um conjunto H de tamanho $\leq b$ que intercepte todos os S_i, se um tal H existir. Em outras palavras, queremos $H \cap S_i \neq \emptyset$ para todo i. Mostre que *Conjunto Incidente* é NP-completo.

7.10 . SUBGRAFO DENSO: dados um grafo G e dois inteiros a e b, encontre um conjunto de a vértices de G tal que exista pelo menos b arestas entre eles. Prove que o problema é NP-completo.

8. Lidando com NP-Completude

8.1 . Um quadrado mágico de ordem 3 é uma tabela 3×3 preenchida com nove números inteiros distintos de 1 a 9, de modo que a soma dos números em cada linha, coluna e duas diagonais de ponta a ponta seja a mesma. **Implemente** um algoritmo backtracking e encontre todos os quadrados mágicos de ordem 3.

8.2 . **Implemente** um algoritmo *backtracking* para o PROBLEMA DO PASSEIO DO CAVALO, iniciando de uma das quinas. Informe o tempo em segundos e apresente a solução.

8.3 . Projete um algoritmo para o PROBLEMA DO CAMINHO HAMILTONIANO de um vértice fixo s.

8.4 . Escolha um problema de otimização e dê um exemplo de uma execução de um algoritmo *branch-and-bound* sobre o problema escolhido. Escolha um instância pequena. Isso implica decidir:

a) Como calcular um limitante para o problema?

b) Como você expandir um nó em subproblemas?

c) Qual subproblema escolher?

8.5 . De forma similar à questão anterior (responda as perguntas!), projete um algoritmo *branch-and-bound* para o problema da COBERTURA DE CONJUNTO.

8.6 . No problema da ÁRVORE DE STEINER MÍNIMA, a entrada consiste em: um grafo completo $G = (V, E)$ com distâncias d_{uv} entre todos os pares de nós; e um determinado conjunto de vértices terminais $V' \subseteq V$. O objetivo é encontrar uma árvore de custo mínimo que inclua os vértices V. Essa árvore pode ou não incluir nós em $V \setminus V'$. Sabe-se que este problema é NP-difícil, proponha uma heurística construtiva para gerar uma solução viável para o problema (note que a solução pode não ser ótima).

8.7 . Apresente o algoritmo de busca local 2-opt para o TSP, dê um exemplo de execução e mostre que ele não é exato.

8.8 . Escolha um problema de otimização NP-difícil e apresente um estrutura de vizinhança para tal problema.

8.9 . Escolha 5 meta-heurístias do livro "Handbook of Metaheuristics" (`https://www.springer.com/gp/book/9783319910857`, cada capítulo fala de uma meta-heurística). Para cada uma das 5, descreva o algoritmo e explique quais os mecanismos de intensificação e diversificação da busca.

8.10 . No problema do BIN PACKING, a entrada consiste em: n itens com tamanhos $s_1, s_2, \ldots, s_n$ em que $s_i \in [0, 1]$. O objetivo é encontrar o menor número de "*bins*" (caixas) unitárias para armazenar os n itens. O algoritmo First Fit consiste em:

```
1: procedure ALGORITMO FIRST FIT(vetor s[1,...,n])
2:     for i ← 1 → n do
3:         Coloque o item i no bin de menor índice que tenha espaço disponível ≥ s_i
4:     end for
5: end procedure
```

Mostre que o algoritmo é 2-aproximado. **Dica**: O FF não deixa, ao final, dois *bins* com espaço utilizado $\leq 0{,}5$.