ANÁLISE DE ALGORITMOS

2º. SEMESTRE DE 2026

EXERCÍCIOS

Entrega. Entregue suas soluções no e-Disciplinas.

Política de colaboração e uso de fontes. Todo trabalho entregue por você deve ser seu. Você é encorajado a discutir com seus colegas o material visto em sala e os enunciados dos exercícios, mas tome cuidado para não compartilhar seu trabalho além do permitido. Todos os exercícios a serem entregues nesta disciplina serão

INDIVIDUAIS.

Nestes exercícios individuais, você não deve compartilhar suas soluções ou mesmo ideias de soluções com seus colegas. Você não deve procurar soluções de terceiros (como de amigos, LLMs, ou na Web). Caso você acidentalmente encontre e leia a solução de algum exercício em algum lugar, você deve citar esta fonte em sua solução. Caso você acidentalmente acabe colaborando com colegas na descoberta de uma solução, você deve citar esta colaboração em sua solução. Seu desempenho nesta disciplina ficará prejudicado caso você viole essas regras.

Fontes dos exercícios. Vários (maioria?) dos exercícios virão de nossa bibliografia principal (CLRS, 3a. edição). Você é encorajado a fazer todos os exercícios e problemas do CLRS relacionados ao material vista na disciplina.

Exercícios para entrega. Os exercícios para entrega estão marcados com uma data de entrega.

Exercícios marcados (*M*). Os exercícios marcados (M) são para os mais matematicamente inclinados, e não serão cobrados nesta disciplina.

* * * * *

E1 Faça o Exercício 3.1-3 de **CLRS**.

E2 Faça o Problema 3-2 de **CLRS**.

E3 Faça o Exercício M01 Mergesort com menos acessos à memória. **{*Data de entrega*: 1/9/2026}**

E4 Faça o Exercício M02 Partição de K&R. **{*Data de entrega*: 1/9/2026}**

E5 (*M*) Considere o algoritmo de partição de K&R, como dado em PARTICIONE(A, p, r). Suponha que $A[1\,..\,N]$ seja uma permutação de $[N] = \{1, \ldots, N\}$, escolhida uniformemente ao acaso. Ao executar PARTICIONE$(A, 1, N+1)$, obtemos duas permutações: uma em $A[1\,..\,q)$ e outra em $A[q+1\,..\,N]$. Prove que a permutação em $A[1\,..\,q)$ é uma permutação de $[q-1]$ sem viés, isto é, todas as permutações de $[q-1]$ ocorrem com a mesma probabilidade em $A[1\,..\,q)$. Analogamente, prove que a permutação em $A[q+1\,..\,N]$ é uma permutação de $\{q+1, \ldots, N\}$ sem viés. Prove que as permutações em $A[1\,..\,q)$ e $A[q+1\,..\,N]$ são independentes uma da outra.

E6 (*M*) Repita o Exercício **E5** considerando o algoritmo de partição de Sedgewick.

Date: Versão de 2026/10/9, 8:50pm.

E7 Considere o algoritmo de partição de K&R, como dado em PARTICIONE(A, p, r). Fixe $1 \le a < b \le N$.

(i) Suponha que $A[1\,..\,N]$ seja uma permutação de $[N] = \{1, \ldots, N\}$, escolhida uniformemente ao acaso. Ao executar PARTICIONE$(A, 1, N+1)$, uma comparação entre a e b pode ou não ocorrer na Linha 4 de PARTICIONE. Qual é a probabilidade de tal comparação acontecer?

(ii) Suponha que $A[1\,..\,N]$ seja uma permutação de $[N] = \{1, \ldots, N\}$. Seja X_{ab} o número de comparações que são feitas entre a e b ao executarmos QUICKSORT$(A, 1, N+1)$. Prove que $X_{ab} \in \{0, 1\}$.

(iii) Suponha que $A[1\,..\,N]$ seja uma permutação de $[N] = \{1, \ldots, N\}$, escolhida uniformemente ao acaso. Prove que $\mathbb{P}(X_{ab} = 1) = 2/(b - a + 1)$, onde X_{ab} é como definido acima. [*Sugestão.* Considere a *primeira* chamada de PARTICIONE em que *algum* elemento de $\{a, a+1, \ldots, b\}$ é o pivô da partição. Note que, se essa chamada for PARTICIONE(A, p, r), então todos os elementos de $\{a, \ldots, b\}$ ocorrem em $A[p, r)$. O que você sabe sobre a ordem dos elementos em $A[p, r)$? Por quê? Basta agora usar (i) acima.]

E8 Descreva um algoritmo determinístico $O(N \log k)$ para determinar um k-ésimo menor elemento de uma lista de N valores. [*Observação.* Em exercícios como este, espera-se que você dê um pseudocódigo para seu algoritmo, prove sua correção e prove que seu consumo de tempo é como exigido.]

E9 Temos uma coleção de N parafusos e N porcas de tamanhos variados. Sabemos que existe um pareamento entre os N parafusos e as N porcas de forma que os elementos pareados encaixam-se perfeitamente. Isto é, se um parafuso p está pareado com uma porca p', então p e p' encaixam-se perfeitamente. Podemos chamar tal pareamento de "pareamento completo" ou "pareamento perfeito", pois todos os N parafusos e todas as N porcas estão pareados.

Para encontrar um tal pareamento, a única operação que podemos fazer é escolher um parafuso p e uma porca p' e experimentar. Há três possíveis resultados para esse experimento: (1) p e p' encaixam-se perfeitamente, (2) p é muito grande para p', (3) p é muito pequeno para p'. Queremos encontrar um pareamento perfeito fazendo "poucos" experimentos como esse. É fácil ver que podemos encontrar um pareamento perfeito executando no máximo N^2 experimentos. Queremos algo bem melhor: *sua tarefa é encontrar um algoritmo linearítmico* (*isto é,* $O(N \log N)$) *para este problema.* Neste exercício, permitimos algoritmos aleatorizados. Ademais, você pode supor que todos os parafusos têm dimensões diferentes e, assim, todas as porcas também têm dimensões diferentes (lembre que estamos supondo que há um pareamento perfeito).

(i) Descreva seu algoritmo de forma precisa, fornecendo pseudocódigos e quaisquer outros detalhes que tornem sua implementação uma tarefa simples.

(ii) Prove que seu algoritmo consome tempo $O(N \log N)$. Se você produzir um algoritmo aleatorizado, basta provar que o consumo de tempo *esperado* é $O(N \log N)$.

{*Data de entrega*: 15/9/2026}

E10 A análise de complexidade do algoritmo SELECT-MOM visto em sala supõe algo sobre a entrada: sem essa hipótese, não é verdade que ele consome tempo $O(N)$ no pior

caso. Descreva um vetor $A[1\,..\,N]$ tal que SELECT-MOM$(A, 1, N+1, \lceil N/2 \rceil)$ consome tempo $\Omega(N^2)$. Por que a prova de linearidade falha para essa entrada?

E11 (Continuação de **E10**) Descreva, em linhas gerais, como alterar SELECT-MOM para que ele seja linear para qualquer instância (A, k). Dê duas soluções. Justifique suas soluções. [*Sugestão.* Para uma de suas soluções, estude o 3-*way partitioning* de Dijkstra (veja a aula de 18/8/2026).] **{*Data de entrega*: 15/9/2026}**

E12 Ao descrevermos o algoritmo de Strassen para multiplicação de matrizes em sala, supusemos que estávamos multiplicando duas matrizes $n \times n$ com n potência de 2. Podemos remover esta hipótese, simplesmente considerando blocos formados por $\lceil n/2 \rceil$ ou $\lfloor n/2 \rfloor$ linhas e $\lceil n/2 \rceil$ ou $\lfloor n/2 \rfloor$ colunas? [*Observação.* Em questões dessa forma (com respostas "sim" ou "não"), você deve, naturalmente, justificar sua resposta.]

E13 Considere o algoritmo de Strassen, como visto em sala, que serve para multiplicar matrizes $n \times n$ com n potência de 2. Como podemos usar este algoritmo para implementar um outro algoritmo que multiplica matrizes $n \times n$ para qualquer $n \geq 1$ com consumo de tempo $O(n^{\lg 7})$?

E14 (PC[1] 111105 Cortes de tora) Você deve cortar uma tora de madeira em vários pedaços. Você solicitará o serviço à *Analog Cutting Machinery* (ACM), que cobra, por corte, um valor igual ao comprimento da tora a ser cortada. A máquina de corte dela permite que apenas um corte seja feito por vez. Se quisermos fazer vários cortes, é fácil ver que ordens diferentes desses cortes podem levar a preços diferentes. Por exemplo, considere uma tora com 10 metros de comprimento, que tem de ser cortada a 2, 4 e 7 metros de uma mesma extremidade. Há várias possibilidades. Podemos primeiro fazer o corte dos 2 metros, depois dos 4 e depois dos 7. Tal ordem custa $10 + 8 + 6 = 24$, porque a primeira tora tinha comprimento 10, o pedaço cortado em seguida tinha comprimento 8 e o último pedaço cortado tinha comprimento 6. Se cortássemos na ordem 4, depois 2, depois 7, pagaríamos $10 + 4 + 6 = 20$, que é o custo mínimo neste exemplo. Descreva um algoritmo que, dado o comprimento L da tora e N pontos de corte $0 < p_1 < \cdots < p_N < L$, encontre o custo mínimo para executar esses cortes na ACM. Seu algoritmo deve também devolver uma ordem de corte que atinja esse custo. Quais são os consumos de tempo e de espaço de seu algoritmo? (Espera-se que seu algoritmo consuma tempo $O(N^3)$ e espaço $O(N^2)$.) **{*Data de entrega*: 22/9/2026}**

E15 (Limite inferior no caso médio) Considere uma árvore de decisão para ordenar $A[1\,..\,n]$, como vista em sala: cada nó interno é rotulado por uma comparação "$A[i] \leq A[j]$?" e tem dois filhos, correspondentes às respostas "sim" e "não"; cada folha (ou "nó externo") é rotulada por uma ordem dos elementos de $A[1\,..\,n]$; e a árvore está correta, isto é, para toda entrada com elementos distintos, a folha atingida dá a ordem crescente dessa entrada. Para cada entrada A, seja $X(A)$ o número de comparações feitas com a entrada A, isto é, a profundidade da folha atingida por A. Suponha que $A[1\,..\,n]$ seja uma permutação de $[n] = \{1, \ldots, n\}$ escolhida uniformemente ao acaso (cada uma das $n!$ permutações ocorre com probabilidade $1/n!$), e seja $\mathbb{E}[X]$ o valor esperado de $X(A)$.

(i) Prove que permutações diferentes atingem folhas diferentes.

[1] Programming Challenges (Skiena & Revilla)

(*ii*) Prove que, em qualquer árvore binária, a soma das profundidades de quaisquer $L \geq 1$ folhas distintas é pelo menos $L \lg L$. [*Sugestão.* A função $f(x) = x \lg x$ é convexa em $(0, \infty)$, isto é, $f(\lambda x + (1-\lambda)y) \leq \lambda f(x) + (1-\lambda) f(y)$ para quaisquer $x, y > 0$ e $0 \leq \lambda \leq 1$: o gráfico de f entre x e y fica abaixo do segmento que liga $(x, f(x))$ a $(y, f(y))$. Em particular, $f(x) + f(y) \geq 2f((x+y)/2)$.]

(*iii*) Conclua que $\mathbb{E}[X] \geq \lg n!$ e que $\mathbb{E}[X] = \Omega(n \lg n)$.

[*Observação.* Isso mostra que todo algoritmo de ordenação baseado em comparações consome tempo $\Omega(n \lg n)$ não só no pior caso, mas também no caso médio.]

E16 (Karatsuba para polinômios) Um polinômio de grau menor que n é dado pelo vetor $P[0..n-1]$ de seus coeficientes: $P(x) = \sum_{i=0}^{n-1} P[i]\, x^i$. O produto de dois tais polinômios P e Q é o polinômio $R = PQ$ dado pelo vetor $R[0..2n-2]$, onde $R[k] = \sum_{i+j=k} P[i]\, Q[j]$. Suponha que os coeficientes sejam números (inteiros, reais ou complexos) e que cada operação aritmética entre coeficientes consome tempo constante. O algoritmo direto faz n^2 multiplicações de coeficientes.

(*i*) Inspirando-se no algoritmo KARATSUBA(X, Y, n) visto em sala, escreva um algoritmo KARATSUBA-POLI(P, Q, n) que devolve PQ. Prove que seu algoritmo está correto.

(*ii*) Seja $M(n)$ o número de multiplicações de coeficientes que KARATSUBA-POLI(P, Q, n) faz. Dê uma recorrência para $M(n)$ e prove que $M(n) = n^{\lg 3}$ quando n é potência de 2. Compare $M(n)$ com n^2 para $n = 2^{10}$.

(*iii*) Prove que KARATSUBA-POLI pode ser implementado de forma a consumir tempo $O(n^{\lg 3})$ para todo $n \geq 1$.

(*iv*) Na linha 7 de KARATSUBA, a chamada recursiva é feita com $\lceil n/2 \rceil + 1$ algarismos. Por que isso é necessário para inteiros, mas não para polinômios?

E17 (Corte de chapas) Chapas retangulares de aço são vendidas em tamanhos inteiros: uma chapa $a \times b$ (largura a, altura b) é vendida por $p[a, b] \geq 0$ reais, onde a tabela $p[1..A, 1..B]$ é dada. Temos uma chapa $A \times B$ e podemos cortá-la antes de vender os pedaços. Um *corte de guilhotina* divide uma chapa $a \times b$ em duas chapas por meio de um corte reto, de um lado ao outro da chapa, paralelo a um dos lados e em posição inteira: obtemos as chapas $i \times b$ e $(a-i) \times b$ (corte vertical, $1 \leq i < a$), ou as chapas $a \times j$ e $a \times (b-j)$ (corte horizontal, $1 \leq j < b$). Os pedaços obtidos podem ser cortados novamente da mesma forma, e não podem ser girados (as chapas $a \times b$ e $b \times a$ podem ter preços diferentes).

(*i*) Seja $q[a, b]$ o maior valor de venda que se pode obter de uma chapa $a \times b$. Dê uma recorrência para $q[a, b]$, análoga à recorrência do corte de hastes, e prove que ela está correta.

(*ii*) Escreva um algoritmo de programação dinâmica que calcula $q[A, B]$ e que permite listar os cortes de um plano de cortes ótimo (como em CORTA-HASTE-BOTTOM-UP-COMPLETO e LISTA-CORTES). Quais são os consumos de tempo e de espaço de seu algoritmo? (Espera-se tempo $O(AB(A+B))$ e espaço $O(AB)$.)

E18 (Parentização com custo geral) Suponha que tenhamos uma operação $*$ entre matrizes tal que, se X é $a \times b$ e Y é $b \times c$, então $X * Y$ é uma matriz $a \times c$, e calcular $X * Y$ custa $f(a, b, c)$, onde f é uma função dada, com valores não negativos, que pode ser calculada em tempo constante. Suponha que $*$ seja associativa: $X * (Y * Z) = (X * Y) * Z$. Considere uma cadeia $A_1, \ldots, A_n$ de matrizes em que A_i é $p_{i-1} \times p_i$, como no problema

da multiplicação iterada de matrizes. O custo de uma parentização de $A_1 * \cdots * A_n$ é a soma dos custos das $n-1$ operações $*$ que ela executa.

(*i*) Seja $m[i,j]$ o custo mínimo de uma parentização de $A_i * \cdots * A_j$. Dê uma recorrência para $m[i,j]$, prove que ela está correta e escreva um algoritmo que consome tempo $O(n^3)$, análogo a MATRIX-CHAIN-ORDER, que determina $m[1,n]$ e uma parentização ótima.

(*ii*) Suponha que $f(a,b,c) = b^2$. Prove que todas as parentizações de $A_1 * \cdots * A_n$ têm o mesmo custo e determine esse custo.

(*iii*) Generalize o item (*ii*) para $f(a,b,c) = g(b)$, onde g é uma função qualquer.

E19 (Todas as sscos máximas) Sejam $X[1\,..\,m]$ e $Y[1\,..\,n]$ sequências de símbolos de um alfabeto A, e seja $c[i,j]$ o comprimento de uma ssco máxima de $X[1\,..\,i]$ e $Y[1\,..\,j]$, como em LCS-LENGTH. Vimos em sala como obter *uma* ssco máxima de X e Y (GET-LCS). Agora queremos listar *todas* as sscos máximas de X e Y, cada uma exatamente uma vez. Duas sscos são consideradas iguais se forem iguais como sequências de símbolos, mesmo que ocorram em posições diferentes de X ou de Y; por exemplo, se $X = \mathrm{A\,A}$ e $Y = \mathrm{A}$, há uma única ssco máxima, a saber, A. Sejam N o número de sscos máximas de X e Y e $L = c[m,n]$ o comprimento delas. Suponha que os símbolos de A possam ser usados como índices de vetores (por exemplo, $A = \{1,\ldots,|A|\}$).

(*i*) Para $0 \le i \le m$ e $a \in A$, seja $\mathrm{ult}_X[i,a]$ o maior $i' \le i$ tal que $X[i'] = a$ (ou 0, se não houver tal i'), e defina $\mathrm{ult}_Y[j,a]$ analogamente. Sejam i e j tais que $k = c[i,j] \ge 1$, e sejam $a \in A$, $i' = \mathrm{ult}_X[i,a]$ e $j' = \mathrm{ult}_Y[j,a]$. Prove que, se $i' \ge 1$, $j' \ge 1$ e $c[i'-1,j'-1] = k-1$, então as sscos máximas de $X[1\,..\,i]$ e $Y[1\,..\,j]$ que terminam em a são exatamente as sequências Wa, onde W é uma ssco máxima de $X[1\,..\,i'-1]$ e $Y[1\,..\,j'-1]$; caso contrário, nenhuma ssco máxima de $X[1\,..\,i]$ e $Y[1\,..\,j]$ termina em a.

(*ii*) Escreva um algoritmo que lista todas as sscos máximas de X e Y, cada uma exatamente uma vez, e prove que ele está correto. Qual é o consumo de tempo de seu algoritmo? (Espera-se tempo $O(mn+|A|(m+n+NL))$; em particular, $O(mn+NL)$ se $|A|$ é constante. Note que NL é o tamanho da saída.)

(*iii*) Mostre que, para todo n par, existem sequências X e Y, ambas de comprimento n, com $2^{n/2}$ sscos máximas. (Assim, listar todas as sscos máximas pode consumir tempo exponencial em n.) [*Observação.* O alfabeto A pode depender de n.]

E20 (Subsequência palindrômica máxima) Uma sequência $Z[1\,..\,k]$ é um *palíndromo* se $Z[t] = Z[k+1-t]$ para $t = 1,\ldots,k$. Dada uma sequência $X[1\,..\,n]$, queremos encontrar uma subsequência de X que seja um palíndromo e tenha comprimento máximo.

(*i*) Para $1 \le i \le j \le n$, seja $c[i,j]$ o comprimento de uma subsequência palindrômica máxima de $X[i\,..\,j]$, e convencione $c[i,i-1] = 0$. Prove que $c[i,i] = 1$ e que, para $i < j$,

$$c[i,j] = \begin{cases} c[i+1,j-1] + 2 & \text{se } X[i] = X[j], \\ \max\{c[i+1,j],\, c[i,j-1]\} & \text{se } X[i] \ne X[j]. \end{cases}$$

(*ii*) Escreva um algoritmo que consome tempo $O(n^2)$, calcula $c[1,n]$ e devolve uma subsequência palindrômica máxima de X (como em GET-LCS).

(*iii*) Seja X^R a sequência X invertida, isto é, $X^R[t] = X[n+1-t]$. Prove que o comprimento de uma ssco máxima de X e X^R é igual a $c[1,n]$. Mostre que uma ssco máxima de X e X^R pode não ser um palíndromo, e explique como obter, a partir de uma ssco máxima de X e X^R, uma subsequência palindrômica de X de mesmo comprimento.

E21 (ABB ótima com buscas sem sucesso) Sejam $x_1 < \cdots < x_n$ as chaves a serem armazenadas numa ABB, e convencione $x_0 = -\infty$ e $x_{n+1} = +\infty$. Uma ABB para essas chaves tem n nós internos, que contêm as chaves, e $n+1$ nós externos (os ponteiros nulos); da esquerda para a direita, os nós externos são numerados $0, 1, \ldots, n$, e o nó externo j é onde terminam as buscas sem sucesso por valores y com $x_j < y < x_{j+1}$. Como em sala, uma busca por x_i faz h_i comparações, onde h_i é o número de nós no caminho da raiz até x_i; uma busca sem sucesso que termina no nó externo j faz h'_j comparações, onde h'_j é o número de nós internos no caminho da raiz até o nó externo j. Suponha que, em cada busca, o valor procurado seja x_i com probabilidade p_i $(1 \le i \le n)$, ou um valor y com $x_j < y < x_{j+1}$ com probabilidade q_j $(0 \le j \le n)$, onde $\sum_{i=1}^n p_i + \sum_{j=0}^n q_j = 1$. O custo esperado de uma ABB é

$$\sum_{i=1}^{n} p_i h_i + \sum_{j=0}^{n} q_j h'_j.$$

(*i*) Para $1 \le i \le j+1 \le n+1$, seja $c[i,j]$ o custo mínimo de uma ABB para as chaves $x_i, \ldots, x_j$ (cujos nós externos são $i-1, \ldots, j$), onde o custo é definido como acima, mas somando apenas sobre essas chaves e esses nós externos, e seja $s[i,j] = \sum_{k=i}^{j} p_k + \sum_{l=i-1}^{j} q_l$. Prove que $c[i,i-1] = 0$ e que, para $i \le j$,

$$c[i,j] = \min_{i \le k \le j} \{c[i,k-1] + c[k+1,j]\} + s[i,j].$$

(*ii*) Escreva um algoritmo que consome tempo $O(n^3)$ e devolve uma ABB ótima.

(*iii*) Para $i \le j$, seja raiz$[i,j]$ o maior k que atinge o mínimo em (*i*). Knuth provou que raiz$[i,j-1] \le$ raiz$[i,j] \le$ raiz$[i+1,j]$ sempre que $i < j$. Usando esse fato (sem prová-lo), mostre como reduzir o consumo de tempo do seu algoritmo para $O(n^2)$.

E22 (Subárvores disjuntas) Seja T uma árvore, isto é, um grafo conexo acíclico, e sejam $T_1, \ldots, T_n$ subárvores de T (subgrafos conexos de T), cada uma dada pela lista de seus vértices. Queremos encontrar índices $i_1 < \cdots < i_m$, com m máximo, tais que as subárvores $T_{i_1}, \ldots, T_{i_m}$ são disjuntas duas a duas nos vértices, isto é, $V(T_{i_a}) \cap V(T_{i_b}) = \emptyset$ sempre que $a \ne b$.

(*i*) Dê um algoritmo para esse problema.

(*ii*) Prove que seu algoritmo está correto.

(*iii*) Qual é o consumo de tempo de seu algoritmo, em função de $|V(T)|$, de n e de $\sum_i |V(T_i)|$?

[*Sugestão.* Os intervalos do problema dos intervalos disjuntos são as subárvores de um caminho. Enraíze T em um vértice qualquer.] **{*Data de entrega*: 23/10/2026}**

E23 (Huffman e entropia) Seja A um alfabeto com $n \ge 2$ símbolos, com "frequências" $f[c] > 0$ $(c \in A)$ tais que $\sum_{c \in A} f[c] = 1$. Seja $w \in A^*$ um texto com N símbolos, no qual cada símbolo c ocorre exatamente $f[c]\,N$ vezes. Como em sala, para uma árvore de código T para A, escrevemos $d_T(c)$ para a profundidade de c em T (o comprimento do

código de c), e $B(T) = \sum_{c \in A} f[c]\, d_T(c)$; assim, w codificado com o código dado por T tem $N\, B(T)$ bits. Seja

$$H(f) = \sum_{c \in A} f[c] \lg \frac{1}{f[c]}$$

a entropia da distribuição f.

(*i*) Prove que $\sum_{c \in A} 2^{-d_T(c)} = 1$ para toda árvore de código T para A.

(*ii*) Prove que, se ℓ_c ($c \in A$) são inteiros positivos com $\sum_{c \in A} 2^{-\ell_c} \leq 1$, então existe um código livre de prefixo cujas palavras têm comprimentos ℓ_c ($c \in A$). Conclua que existe uma árvore de código T para A com $d_T(c) \leq \ell_c$ para todo c.

(*iii*) Prove que w codificado com o código de Huffman para f tem menos que $N(H(f)+1)$ bits. [*Sugestão.* Considere $\ell_c = \lceil \lg(1/f[c]) \rceil$.]

(*iv*) Prove que $B(T) \geq H(f)$ para toda árvore de código T para A e, portanto, que w codificado com qualquer código livre de prefixo tem pelo menos $N\, H(f)$ bits. [*Sugestão.* Estime $H(f) - B(T)$ por cima, escrevendo-a como uma única soma, e use que $\ln x \leq x - 1$ para todo $x > 0$.]

(*v*) Conclua que o número b de bits de w codificado com o código de Huffman para f satisfaz $N\, H(f) \leq b < N(H(f) + 1)$.

E24 (Tarefas unitárias com prazos) Temos n tarefas, todas com tempo de processamento $t_i = 1$, e com prazos d_i inteiros positivos ($1 \leq i \leq n$). Como em sala, um escalonamento é uma permutação π de $1, \ldots, n$, e a tarefa $\pi[k]$ é executada no intervalo de tempo $[k-1, k)$; assim, $f_{\pi[k]} = k$. Dizemos que a tarefa i é *pontual* em π se $f_i \leq d_i$. Queremos um escalonamento com o maior número possível de tarefas pontuais.

(*i*) Dê um algoritmo guloso que resolve esse problema em tempo $O(n \lg n)$, e prove que ele está correto.

(*ii*) Mostre que podemos supor que $d_i \leq n$ para todo i, e obtenha um algoritmo que consome tempo $O(n)$.

E25 (Fila de prioridade com chaves pequenas) Seja $C \geq 2$ um inteiro. Queremos uma fila de prioridade Q cujos elementos são inteiros em $\{1, \ldots, n\}$, cada elemento x com uma chave $\text{key}[x] \in \{1, \ldots, C\}$, com as operações

- INSERT(Q, x, k): insere em Q o elemento x (que não está em Q) com chave k;
- EXTRACT-MIN(Q): remove de Q e devolve um elemento de chave mínima;
- DECREASE-KEY(Q, x, k): muda a chave do elemento x de Q para $k \leq \text{key}[x]$.

Mostre como implementar Q de modo que cada uma dessas operações consuma tempo $O(\lg C)$ e que a criação de uma fila vazia consuma tempo $O(n + C)$. [*Sugestão.* Use um vetor de C listas ("baldes"), uma para cada valor de chave, e uma árvore binária completa cujas folhas são os baldes.]

E26 (Prim com custos pequenos) Seja G um grafo conexo com n vértices e m arestas, com custos inteiros $c(uv) \in \{1, \ldots, C\}$ nas arestas, onde $C \geq 2$. Mostre como implementar PRIM(G, c) usando a fila do Exercício **E25** de modo que o consumo de tempo seja $O(m \lg C + n + C)$. Note que, em PRIM, os valores iniciais $\text{key}[v] = \infty$ não estão em $\{1, \ldots, C\}$; explique como contornar isso.

E27 (Todas as subsequências crescentes máximas) Seja $c_1, \ldots, c_n$ uma sequência de números distintos. Considere o algoritmo PATIENCE visto em sala: cada c_i é empilhado

na pilha S_j mais à esquerda cujo topo é maior que c_i (ou numa nova pilha, se não há tal S_j). Uma LIS de $c_1, \ldots, c_n$ é uma subsequência crescente de comprimento máximo.

(*i*) Mostre que, para todo n par, existe uma sequência de n números distintos com $2^{n/2}$ LISs. (Assim, listar todas as LISs pode consumir tempo exponencial em n.)

(*ii*) Suponha que, ao empilharmos c_i em S_j, com $j \geq 2$, guardemos referências para *todos* os elementos que estão em S_{j-1} naquele momento e que são menores que c_i. Prove que esses elementos formam um segmento não vazio no topo de S_{j-1}, de modo que basta guardar dois índices, e que isso pode ser feito sem alterar o consumo de tempo $O(n \lg n)$ de PATIENCE.

(*iii*) Prove que as LISs de $c_1, \ldots, c_n$ são exatamente as sequências obtidas começando em um elemento da última pilha não vazia e seguindo referências como em (*ii*) até chegar a S_1, e que cada LIS é obtida exatamente uma vez. [*Sugestão.* Prove que c_i está em S_j se e somente se j é o comprimento de uma subsequência crescente máxima que termina em c_i.]

(*iv*) Dê um algoritmo que conta as LISs de $c_1, \ldots, c_n$ com $O(n \lg n)$ operações (supondo que operações aritméticas consumam tempo constante).

(*v*) Dê um algoritmo que lista todas as LISs de $c_1, \ldots, c_n$ em tempo $O(n \lg n + NL)$, onde N é o número de LISs e L é o comprimento delas.

{*Data de entrega*: 23/10/2026}