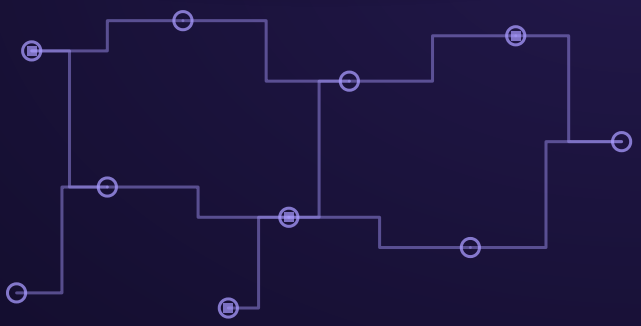
TECNOLOGIA DA INFORMAÇÃO

SÉRIE TECNOLOGIA PAULISTA

COORDENAÇÃO E REVISÃO

Emerson Braga de Souza



PREPARATÓRIO CETESB ANALISTA DE TI

Concurso 2027 · Tecnologia da Informação – Sistemas

CURSO LIVRE ICEAD

Material do aluno

Capítulo 9 · Representação de dados, lógica de programação, complexidade, estruturas de dados, ordenação, pesquisa e hashing

Tecnologia da Informação · capítulo 9 de 11

Edital nº 01/2026 · prova em 21/02/2027
Atualizado até 20/11/2026

ICEAD
INSTITUTO DE CURSOS EAD

EXPEDIENTE

Curso Preparatório CETESB – Analista de Tecnologia da Informação 2027

Capítulo 9 · Representação de dados, lógica de programação, complexidade, estruturas de dados, ordenação, pesquisa e hashing

OBRA COLETIVA

Organizada e publicada pela ICEAD – Instituto de Cursos EAD (Lei nº 9.610/1998, art. 5º, VIII, h, e art. 17)

COORDENAÇÃO E REVISÃO

Emerson Braga de Souza, professor, advogado (OAB/SP), contador (CRC-SP) e administrador (CRA-SP)

ELABORAÇÃO

Equipe editorial da ICEAD, com apoio de ferramentas de inteligência artificial

COMO ESTE MATERIAL FOI FEITO

Cada capítulo foi escrito a partir das fontes indicadas abaixo e no fim de cada capítulo, conferido por uma segunda revisão independente (conteúdo, contas e gabaritos) e aprovada pela coordenação antes da publicação. As questões são inéditas, escritas para este curso; nenhuma reproduz questão de prova anterior.

FONTES PRINCIPAIS

Edital nº 01/2026 de Abertura de Inscrições para Empregos de Nível Superior do concurso público da Companhia Ambiental do Estado de São Paulo (CETESB), organizado pela Fundação Carlos Chagas, Anexo III (conteúdo programático); documentação técnica e bibliografia de computação indicadas em cada capítulo.

EDIÇÃO

Versão de 20/11/2026. Quando houver correção ou atualização, a versão nova fica disponível na Sala de aula.

ENCONTROU UM ERRO?

Escreva para contato@icead.com.br. Toda correção é registrada e entra na próxima versão.

Curso livre da ICEAD, independente, sem vínculo com a Companhia Ambiental do Estado de São Paulo (CETESB) nem com a Fundação Carlos Chagas. O concurso e as suas regras são definidos só pelo edital oficial e pelas suas retificações.

AVISO LEGAL

Direitos autorais e licença de uso

© 2026 ICEAD, marca de Arizona Marcas e Patentes Ltda. Todos os direitos reservados.

Este material é protegido pela Lei nº 9.610/1998 (Direitos Autorais). A compra dá ao aluno uma **licença de uso pessoal, individual e intransferível**. É proibido reproduzir, distribuir, revender, emprestar ou disponibilizar este arquivo, no todo ou em parte, por qualquer meio.

O compartilhamento não autorizado e a pirataria configuram **crime de violação de direito autoral (Código Penal, art. 184)**, sem prejuízo da responsabilidade civil por perdas e danos.

Exemplar licenciado ao aluno identificado no ato da compra. Cada arquivo pode conter marcação, visível ou oculta, que permite rastrear a origem de cópias irregulares.

Dúvidas: contato@icead.com.br

Sumário

Capítulo 9. Representação de dados, lógica de programação, complexidade, estruturas de dados, ordenação, pesquisa e hashing

1. Representação de dados numéricos
2. Lógica de programação
3. Complexidade de algoritmos
4. Estruturas de dados lineares
5. Árvores
6. Ordenação
7. Pesquisa e hashing
8. Para a prova
9. Treino do capítulo (estilo FCC)

Fontes

Tecnologia da Informação

Capítulo 9. Representação de dados, lógica de programação, complexidade, estruturas de dados, ordenação, pesquisa e hashing

> **O que o edital pede (Edital nº 01/2026, Anexo III, Analista de Tecnologia da Informação – Sistemas):** "Linguagens e Técnicas de Programação: Tipos e representações de dados numéricos; Lógica de Programação. Algoritmos e estrutura de dados: Complexidade de algoritmos; Listas e Pilhas; Vetores e matrizes; Estruturas em árvores; Árvores balanceadas; Métodos de ordenação; Pesquisa e hashing."

Este capítulo foi escrito para o programa de Sistemas da companhia ambiental paulista, que cobra a base clássica da ciência da computação. A FCC pergunta **qual estrutura atende a um requisito**, **qual a complexidade** de um algoritmo e **o que um trecho de pseudocódigo produz**. Os Conhecimentos Específicos têm **30 questões com peso 2** (cap. 7, item 7.1).

1. Representação de dados numéricos

- **Bases:** decimal (10), **binária** (2), octal (8) e **hexadecimal** (16, dígitos 0 a 9 e A a F). Cada dígito hexadecimal corresponde a **4 bits**: 0xFF = 1111 1111 = 255.
- **Conversão binário → decimal:** soma das potências de 2. $1011_2 = 8 + 0 + 2 + 1 =$ **11**.
- **Inteiros com sinal em complemento de dois:** com n bits, o intervalo é de −2^(n−1) a 2^(n−1) − 1. Com **8 bits: −128 a 127**. Para obter −x: inverta os bits de x e some 1. Exemplo: 5 = 0000 0101 → inverte = 1111 1010 → +1 = **1111 1011** (−5).
- **Estouro (*overflow*):** em 8 bits com sinal, 127 + 1 "dá a volta" e resulta em −128.
- **Ponto flutuante (IEEE 754):** sinal, expoente e mantissa (precisão simples: 1, 8 e 23 bits; dupla: 1, 11 e 52 bits). Muitos decimais **não têm representação exata** em binário (0,1, por exemplo): por isso não se compara igualdade de *floats* diretamente nem se usa ponto flutuante para valores monetários (use tipos decimais).

2. Lógica de programação

- **Estruturas:** sequência, **decisão** (se/senão, escolha-caso) e **repetição** (enquanto: testa antes; repita-até: testa depois e executa pelo menos uma vez; para: contador).
- **Teste de mesa:** acompanhe o valor das variáveis a cada linha, como no exemplo abaixo, que soma os ímpares de 1 a 5 (s = 1 + 3 + 5 = **9**).

```
s ← 0; i ← 1
enquanto i ≤ 5 faça
    se i mod 2 = 1 então s ← s + i
```

```
    i ← i + 1
fim
```

- **Operadores lógicos e curto-circuito:** em `A e B`, se A é falso, B não é avaliado; em `A ou B`, se A é verdadeiro, B não é avaliado.
- **Recursão:** função que chama a si mesma, com **caso base** obrigatório. Fatorial(4) = 4 × 3 × 2 × 1 = 24. Sem caso base, estouro da pilha de chamadas.

3. Complexidade de algoritmos

A notação **O** (grande O) descreve o crescimento do custo (tempo ou memória) em função do tamanho n da entrada, no pior caso, desprezando constantes.

Ordem	Nome	Exemplo
O(1)	constante	acesso a vetor por índice
O(log n)	logarítmica	**busca binária**
O(n)	linear	busca sequencial
O(n log n)	linearítmica	merge sort, heap sort
$O(n^2)$	quadrática	dois laços aninhados sobre n (bubble sort)
O(2^n)	exponencial	testar todos os subconjuntos

Regras: laços em **sequência** somam (O(n) + O(n) = O(n)); laços **aninhados** multiplicam (O(n) × O(n) = $O(n^2)$); um laço que **divide** o problema pela metade a cada passo é O(log n).

4. Estruturas de dados lineares

Estrutura	Característica	Operações
Vetor (*array*)	elementos contíguos, tamanho fixo, acesso por índice em O(1)	inserção no meio exige deslocamento, O(n)
Matriz	vetor de vetores (linhas × colunas); em C, armazenada linha a linha	elemento [i][j]
Lista encadeada	nós com dado e ponteiro para o próximo (simples) ou também para o anterior (dupla)	inserção e remoção em O(1) com o ponteiro do lugar; acesso por posição em O(n)
Pilha	**LIFO** (último a entrar, primeiro a sair)	*push*, *pop*, *top*; usada em chamadas de funções, desfazer, avaliação de expressões
Fila	**FIFO** (primeiro a entrar, primeiro a sair)	*enqueue*, *dequeue*; usada em escalonamento e filas de impressão
Fila de prioridade	sai primeiro o de maior prioridade	implementada com *heap*

Exemplo de pilha: empilhar 1, 2, 3; desempilhar; empilhar 4; desempilhar duas vezes. Saem 3, 4 e 2; fica 1.

5. Árvores

- **Conceitos:** raiz, nó, filho, folha (sem filhos), **altura** (maior caminho da raiz a uma folha), grau.
- **Árvore binária:** cada nó tem no máximo dois filhos. **Percursos:** pré-ordem (raiz, esquerda, direita), **em ordem** (esquerda, raiz, direita) e pós-ordem (esquerda, direita, raiz). Em largura (nível a nível) usa uma fila.
- **Árvore binária de busca (ABB):** à esquerda, valores menores; à direita, maiores. O percurso **em ordem** lista os valores **ordenados**. Busca em O(log n) se a árvore estiver equilibrada, mas **O(n)** no pior caso (inserção em ordem crescente gera uma "lista").
- **Árvores balanceadas: AVL** mantém, em cada nó, **fator de balanceamento** (altura da subárvore esquerda menos a da direita) entre −1 e +1, corrigido por **rotações** simples e duplas, garantindo O(log n). **Rubro-negra** usa regras de cores, com balanceamento menos rígido. **Árvore B e B+**: muitos filhos por nó, baixa altura, usadas em **índices de bancos de dados** e sistemas de arquivos.
- ***Heap* binário:** árvore completa em que o pai é maior (*max-heap*) ou menor (*min-heap*) que os filhos; base do heap sort e da fila de prioridade.

6. Ordenação

Método	Pior caso	Melhor caso	Estável?	Observação
Bubble sort	$O(n^2)$	O(n) com verificação de trocas	sim	compara vizinhos e troca
Selection sort	$O(n^2)$	$O(n^2)$	não	seleciona o menor e coloca na posição
Insertion sort	$O(n^2)$	**O(n)** (vetor quase ordenado)	sim	bom para vetores pequenos ou quase ordenados
Merge sort	**O(n log n)**	O(n log n)	**sim**	divide e conquista; usa memória auxiliar O(n)
Quick sort	**$O(n^2)$** (pivô ruim)	O(n log n)	não	O(n log n) em média; ordena no próprio vetor
Heap sort	O(n log n)	O(n log n)	não	usa *heap*; não precisa de memória auxiliar

Estabilidade: um método é estável se mantém a ordem relativa de elementos com chaves iguais.

7. Pesquisa e hashing

- **Busca sequencial:** O(n); funciona em dados desordenados.
- **Busca binária:** exige **dados ordenados** e acesso por índice; compara com o elemento do meio e descarta metade a cada passo: **O(log n)**. Em 1.024 elementos, no máximo cerca de 10 comparações (2^10 = 1.024).