



ESTRUTURAS DE DADOS

EXERCÍCIOS DE IMPLEMENTAÇÃO, CONCEITO E ALGORITMOS

LISTA DE EXERCÍCIOS

BALANÇO TÉCNICO

TIPOS ABSTRATOS DE DADOS

- ① Implemente um TAD **Ponto** que represente um ponto no plano cartesiano com coordenadas (x, y) .

O TAD deve fornecer as seguintes operações:

- Criar um ponto com coordenadas x e y
- Obter as coordenadas x e y
- Calcular a distância do ponto até a origem $(0, 0)$
- Calcular a distância entre dois pontos
- Exibir o ponto no formato “ (x, y) ”

Este exercício introduz conceitos fundamentais de TAD: encapsulamento de dados e operações associadas.

Exemplo de uso

```
Ponto p1 = criarPonto(3, 4);  
Ponto p2 = criarPonto(0, 0);
```

```
Distância até origem: 5.00  
Distância entre p1 e p2: 5.00  
Coordenadas: (3, 4)
```

- ② Implemente um TAD **Data** que represente uma data com dia, mês e ano.

O TAD deve fornecer as seguintes operações:

- Criar uma data (com validação de data válida)
- Verificar se o ano é bissexto
- Comparar duas datas (anterior, igual ou posterior)
- Calcular a diferença em dias entre duas datas
- Adicionar N dias a uma data
- Exibir a data no formato “DD/MM/AAAA”

Este exercício trabalha validação, lógica de calendário e operações aritméticas.

Exemplo de uso

```
Data d1 = criarData(15, 3, 2024);  
Data d2 = criarData(20, 3, 2024);
```

Data válida: true

Bissexto: true

Comparação: d1 é anterior a d2

Diferença: 5 dias

d1 + 10 dias: 25/03/2024

- 3 Implemente um TAD Conjunto que represente um conjunto matemático de inteiros.

O TAD deve fornecer as seguintes operações:

- Criar conjunto vazio
- Inserir elemento (sem repetições)
- Remover elemento
- Verificar se elemento pertence ao conjunto
- Obter tamanho do conjunto
- União de dois conjuntos
- Interseção de dois conjuntos
- Diferença entre dois conjuntos
- Verificar se um conjunto é subconjunto de outro
- Exibir elementos do conjunto

Este exercício trabalha estruturas dinâmicas e operações matemáticas sobre coleções.

Exemplo de uso

Conjunto A = {1, 2, 3, 4}

Conjunto B = {3, 4, 5, 6}

$A \cup B = \{1, 2, 3, 4, 5, 6\}$

$A \cap B = \{3, 4\}$

$A - B = \{1, 2\}$

$3 \in A$: true

$A \subseteq B$: false

- 4 Implemente um TAD **MatrizEsparsa** que represente uma matriz esparsa (com muitos elementos zero) de forma eficiente.

O TAD deve armazenar apenas elementos não-nulos usando lista encadeada e fornecer:

- Criar matriz esparsa com dimensões $M \times N$
- Inserir elemento em posição (i, j)
- Acessar elemento em posição (i, j)
- Somar duas matrizes esparsas
- Multiplicar duas matrizes esparsas
- Transpor a matriz
- Exibir matriz completa (incluindo zeros)

Este exercício avançado trabalha eficiência de memória, estruturas encadeadas e algoritmos de matriz.

Exemplo de uso

```
MatrizEsparsa M1(5, 5);  
M1.inserir(0, 0, 10);  
M1.inserir(2, 3, 20);  
M1.inserir(4, 4, 30);
```

Elementos armazenados: 3 (de 25 possíveis)

```
M1[2][3] = 20  
M1[1][1] = 0
```

Matriz transposta:

```
10  0  0  0  0  
 0  0  0  0  0  
 0  0  0 20  0  
 0  0  0  0  0  
 0  0  0  0 30
```

LISTAS E ARRAYS

- 5 Crie um programa em C++ que implemente uma lista encadeada simples (singly linked list) e exiba seus elementos. O programa deve permitir ao usuário inserir valores na lista e, ao final, exibir todos os elementos na ordem em que foram inseridos.

Este exercício introduz os conceitos fundamentais de listas encadeadas: criação de nós, inserção e travessia da lista.

Exemplo de entrada

```
Digite a quantidade de elementos: 6
Digite o elemento 1: 11
Digite o elemento 2: 9
Digite o elemento 3: 7
Digite o elemento 4: 5
Digite o elemento 5: 3
Digite o elemento 6: 1
```

Exemplo de saída

```
Lista encadeada: 11 9 7 5 3 1
```

- 6 Crie um programa em C++ que crie uma lista encadeada simples de n nós e exiba seus elementos em ordem reversa. O programa deve solicitar ao usuário a quantidade de elementos e seus valores, e depois exibir a lista do último elemento para o primeiro.

Este exercício trabalha a travessia recursiva ou iterativa de listas encadeadas em ordem reversa.

Exemplo de entrada

```
Digite a quantidade de elementos: 6
Digite o elemento 1: 11
Digite o elemento 2: 9
Digite o elemento 3: 7
Digite o elemento 4: 5
Digite o elemento 5: 3
Digite o elemento 6: 1
```

Exemplo de saída

```
Lista original: 11 9 7 5 3 1
Lista reversa: 1 3 5 7 9 11
```

- 7 Crie um programa em C++ que crie uma lista encadeada simples de n nós e conte o número total de nós. O programa deve permitir inserir elementos na lista e, ao final, exibir a quantidade total de nós presentes.

Este exercício introduz a operação de contagem de elementos em uma lista encadeada.

Exemplo de entrada

```
Digite a quantidade de elementos: 7
Digite o elemento 1: 13
Digite o elemento 2: 11
Digite o elemento 3: 9
Digite o elemento 4: 7
Digite o elemento 5: 5
Digite o elemento 6: 3
Digite o elemento 7: 1
```

Exemplo de saída

```
Lista encadeada: 13 11 9 7 5 3 1
Número de nós na lista: 7
```

- 8 Crie um programa em C++ que insira um novo nó no início de uma lista encadeada simples. O programa deve criar uma lista com alguns elementos, solicitar um novo valor ao usuário e inseri-lo no início da lista.

Este exercício trabalha a inserção no início da lista, atualizando o ponteiro da cabeça (head).

Exemplo de entrada

Lista original: 13 11 9 7 5 3 1
Digite o valor a ser inserido no início: 0

Exemplo de saída

Lista após inserção no início: 0 13 11 9 7 5 3 1

- 9 Crie um programa em C++ que insira um novo nó no final de uma lista encadeada simples. O programa deve criar uma lista com alguns elementos, solicitar um novo valor ao usuário e inseri-lo no final da lista.

Este exercício trabalha a inserção no final da lista, percorrendo até o último nó.

Exemplo de entrada

Lista original: 13 11 9 7 5 3 1
Digite o valor a ser inserido no final: 0

Exemplo de saída

Lista após inserção no final: 13 11 9 7 5 3 1 0

- ⑩ Crie um programa em C++ que encontre o elemento do meio de uma lista encadeada. O programa deve criar uma lista e determinar qual elemento está na posição central. Se a lista tiver número par de elementos, retorne o segundo elemento do meio.

Este exercício introduz o conceito de dois ponteiros (slow e fast) para encontrar o meio da lista em uma única travessia.

Exemplo de entrada 1

Lista: 7 5 3 1

Exemplo de saída 1

Elemento do meio: 3

Exemplo de entrada 2

Lista: 9 7 5 3 1

Exemplo de saída 2

Elemento do meio: 5

- ⑪ Crie um programa em C++ que insira um novo nó no meio de uma lista encadeada simples. O programa deve criar uma lista, calcular a posição do meio e inserir o novo valor nessa posição.

Este exercício combina a busca pelo elemento do meio com a inserção em posição específica.

Exemplo de entrada

Lista original: 7 5 3 1

Digite o valor a ser inserido no meio: 9

Exemplo de saída (após primeira inserção)

Lista após inserção: 7 5 9 3 1

Exemplo de saída (após segunda inserção)

Lista após inserção: 7 5 9 11 3 1

- 12 Crie um programa em C++ que obtenha o n-ésimo nó de uma lista encadeada simples. O programa deve criar uma lista, solicitar ao usuário uma posição e retornar o valor do nó nessa posição (1-indexed).

Este exercício trabalha o acesso por índice em listas encadeadas.

Exemplo de entrada

Lista: 7 5 3 1
Digite a posição desejada: 3

Exemplo de saída

Valor na posição 3: 3

- 13 Crie um programa em C++ que insira um novo nó em qualquer posição de uma lista encadeada simples. O programa deve criar uma lista, solicitar ao usuário a posição e o valor a ser inserido, e inserir o nó na posição especificada (1-indexed).

Este exercício trabalha a inserção em posição arbitrária, tratando casos especiais como inserção no início e no final.

Exemplo de entrada 1

Lista original: 7 5 3 1
Digite a posição: 1
Digite o valor: 12

Exemplo de saída 1

Lista atualizada: 12 7 5 3 1

Exemplo de entrada 2

Lista atual: 12 7 5 3 1
Digite a posição: 4
Digite o valor: 14

Exemplo de saída 2

Lista atualizada: 12 7 5 14 3 1

- 14 Crie um programa em C++ que delete o primeiro nó de uma lista encadeada simples. O programa deve criar uma lista, exibi-la, remover o primeiro elemento e exibir a lista atualizada.

Este exercício trabalha a remoção no início da lista, atualizando o ponteiro da cabeça (head).

Exemplo de entrada

Lista original: 13 11 9 7 5 3 1

Exemplo de saída

Lista após remoção do primeiro nó: 11 9 7 5 3 1

- 15 Crie um programa em C++ que delete um nó do meio de uma lista encadeada simples. O programa deve criar uma lista, calcular a posição do meio, remover o nó dessa posição e exibir a lista atualizada. O processo pode ser repetido até que reste apenas um elemento.

Este exercício combina a busca pelo elemento do meio com a remoção em posição específica.

Exemplo de entrada

Lista original: 9 7 5 3 1

Exemplo de saída (após primeira remoção)

Lista após remover o meio: 9 7 3 1

Exemplo de saída (após segunda remoção)

Lista após remover o meio: 9 7 1

Exemplo de saída (após terceira remoção)

Lista após remover o meio: 9 1

Exemplo de saída (após quarta remoção)

Lista após remover o meio: 9

- 16 Crie um programa em C++ que delete o último nó de uma lista encadeada simples. O programa deve criar uma lista, exibi-la, remover o último elemento e exibir a lista atualizada. O processo pode ser repetido múltiplas vezes.

Este exercício trabalha a remoção no final da lista, percorrendo até o penúltimo nó.

Exemplo de entrada

Lista original: 7 5 3 1

Exemplo de saída (após primeira remoção)

Lista após remoção do último nó: 7 5 3

Exemplo de saída (após segunda remoção)

Lista após remoção do último nó: 7 5

- 17 Crie um programa em C++ que implemente um array dinâmico simples com redimensionamento automático. O programa deve criar uma classe que armazena elementos em um array, e quando o array está cheio, cria um novo array com o dobro do tamanho, copia os elementos e libera a memória antiga.

Este exercício demonstra o mecanismo fundamental de redimensionamento de arrays dinâmicos: realocação e cópia de elementos.

Exemplo de entrada

Digite os elementos (0 para parar):

Elemento 1: 10

Elemento 2: 20

Elemento 3: 30

Elemento 4: 40

Elemento 5: 50

Elemento 6: 0

Exemplo de saída

Capacidade inicial: 4

Redimensionando: capacidade 4 -> 8

Array final: 10 20 30 40 50

Tamanho: 5

Capacidade: 8

- 18 Crie um programa em C++ que implemente uma classe Vector com comportamento similar à lista do Python. A classe deve suportar: adicionar elemento no final (append), remover e retornar último elemento (pop), inserir em posição específica (insert), remover primeira ocorrência de valor (remove), acesso por índice (operator[]), e redimensionamento automático (dobrar quando cheio, reduzir pela metade quando muito vazio).

Este exercício implementa uma estrutura de dados completa com todas as operações básicas de uma lista dinâmica.

Exemplo de interação

```
Vector v;  
v.append(10);  
v.append(20);  
v.append(30);  
v.display();           // [10, 20, 30]  
v.insert(1, 15);       // [10, 15, 20, 30]  
v.remove(20);          // [10, 15, 30]  
int x = v.pop();        // x = 30, v = [10, 15]  
v.display();           // [10, 15]  
v[0] = 100;            // [100, 15]
```

Exemplo de saída

```
Tamanho: 2  
Capacidade: 4  
Elementos: 100 15
```

- 19 Crie um programa em C++ que implemente um sistema de Undo/Redo usando listas duplamente ligadas. O sistema deve permitir executar ações (com descrição), desfazer a última ação (undo) e refazer ações desfeitas (redo). Use duas listas: uma para ações executadas e outra para ações desfeitas.

Este exercício aplica listas duplamente ligadas em um cenário prático de histórico de operações, demonstrando navegação bidirecional.

Exemplo de interação

```
> add 10
Ação: Adicionar 10
> add 20
Ação: Adicionar 20
> multiply 2
Ação: Multiplicar por 2
> undo
Desfeito: Multiplicar por 2
> undo
Desfeito: Adicionar 20
> redo
Refeito: Adicionar 20
> display
Ações ativas: Adicionar 10, Adicionar 20
```

Exemplo de saída

```
Histórico de ações:
[Adicionar 10] <- [Adicionar 20] <- atual
Ações desfeitas: [Multiplicar por 2]
```

FILAS

- 20 Crie um programa em C++ que implemente a operação de enqueue (enfileirar) em uma fila usando array. O programa deve permitir ao usuário inserir elementos na fila até atingir a capacidade máxima, exibindo mensagens apropriadas.

Este exercício introduz o conceito fundamental de inserção em filas (FIFO - First In, First Out) e o gerenciamento do índice de inserção (rear).

Exemplo de entrada

```
Capacidade da fila: 5
Digite os elementos (0 para parar):
Elemento: 10
Elemento: 20
Elemento: 30
Elemento: 0
```

Exemplo de saída

```
Fila: [10, 20, 30]
Tamanho: 3
Capacidade: 5
```

- 21 Crie um programa em C++ que implemente a operação de dequeue (desenfileirar) em uma fila usando array. O programa deve permitir remover elementos do início da fila e exibir o elemento removido. Trate o caso de fila vazia.

Este exercício trabalha a remoção do início da fila e o gerenciamento do índice de remoção (front).

Exemplo de entrada

Fila inicial: [10, 20, 30, 40, 50]
Quantos elementos remover? 3

Exemplo de saída

Removido: 10
Removido: 20
Removido: 30
Fila atual: [40, 50]

- 22 Crie um programa em C++ que implemente a operação de front (consultar início) em uma fila. O programa deve permitir visualizar o elemento do início da fila sem removê-lo, útil para verificar qual elemento será processado a seguir.

Este exercício introduz a operação de consulta que não altera a estrutura da fila.

Exemplo de entrada

Fila: [15, 25, 35, 45]

Exemplo de saída

Elemento do início: 15
Próximo a ser atendido: 15
Fila permanece: [15, 25, 35, 45]

- 23 Crie um programa em C++ que implemente as operações de size (tamanho) e isEmpty (verificar se vazia) em uma fila. O programa deve fornecer métodos para consultar quantos elementos estão na fila e verificar se ela está vazia antes de operações de remoção.

Este exercício trabalha o gerenciamento do estado da fila e validações importantes para evitar erros.

Exemplo de interação

```
Fila vazia? Sim
Inserindo: 10, 20, 30
Tamanho: 3
Fila vazia? Não
Removendo todos...
Fila vazia? Sim
Tentando remover de fila vazia: ERRO - Fila vazia!
```

- 24 Crie um programa em C++ que implemente uma fila completa usando lista encadeada. Ao contrário da implementação com array, a fila deve crescer dinamicamente conforme necessário, sem limite pré-definido de capacidade.

Este exercício demonstra a implementação de fila com alocação dinâmica, utilizando ponteiros para o início (front) e fim (rear) da lista.

Exemplo de entrada

```
Digite elementos (0 para parar):
Elemento: 5
Elemento: 10
Elemento: 15
Elemento: 20
Elemento: 0
```

Exemplo de saída

```
Fila: 5 -> 10 -> 15 -> 20
Tamanho: 4
Desenfileirando: 5
Fila: 10 -> 15 -> 20
```

- 25 Crie um programa em C++ que simule um sistema de atendimento bancário com múltiplos caixas. Clientes chegam e são atendidos em ordem de chegada (FIFO) pelos caixas disponíveis. O sistema deve calcular métricas como tempo médio de espera na fila e tempo médio de atendimento por caixa.

Cada cliente tem um tempo de atendimento aleatório ou pré-definido. O sistema processa clientes até que todos sejam atendidos, distribuindo-os entre os caixas disponíveis.

Este exercício aplica filas em um cenário realista de sistemas de atendimento, demonstrando balanceamento de carga e análise de desempenho.

Exemplo de entrada

```
Número de caixas: 3
Número de clientes: 8
Tempo de atendimento do cliente 1: 5 min
Tempo de atendimento do cliente 2: 3 min
Tempo de atendimento do cliente 3: 8 min
Tempo de atendimento do cliente 4: 2 min
Tempo de atendimento do cliente 5: 6 min
Tempo de atendimento do cliente 6: 4 min
Tempo de atendimento do cliente 7: 7 min
Tempo de atendimento do cliente 8: 3 min
```

Exemplo de saída

```
Distribuição de atendimento:
Caixa 1: Clientes 1(5min), 4(2min), 7(7min) - Total: 14min
Caixa 2: Clientes 2(3min), 5(6min), 8(3min) - Total: 12min
Caixa 3: Clientes 3(8min), 6(4min) - Total: 12min
```

```
Estatísticas:
Tempo médio de espera na fila: 4.25 min
Tempo médio de atendimento: 4.75 min
Caixa mais ocupado: Caixa 1 (14 min)
Total de clientes atendidos: 8
```

- 26 Crie um programa em C++ que implemente um buffer circular (ring buffer) para comunicação entre processos produtor e consumidor. O buffer tem capacidade fixa e opera com política FIFO circular: quando cheio, o produtor espera; quando vazio, o consumidor espera.

O programa deve simular a produção e consumo de dados, mostrando o estado do buffer a cada operação e como os índices de inserção e remoção circulam pelo array.

Este exercício demonstra a eficiência da fila circular em cenários de streaming e comunicação entre processos, eliminando a necessidade de deslocar elementos.

Exemplo de simulação

Capacidade do buffer: 4

Produzir: A

Buffer: [A, _, _, _] (in=1, out=0)

Produzir: B

Buffer: [A, B, _, _] (in=2, out=0)

Consumir: A

Buffer: [_, B, _, _] (in=2, out=1)

Produzir: C

Buffer: [_, B, C, _] (in=3, out=1)

Produzir: D

Buffer: [_, B, C, D] (in=0, out=1) // circular!

Produzir: E

Buffer: [E, B, C, D] (in=1, out=1) // sobrescreve? Não, espera!

Consumir: B

Buffer: [E, _, C, D] (in=1, out=2)

Produzir: E

Buffer: [E, E, C, D] (in=2, out=2)

Exemplo de saída

Total produzido: 5 itens

Total consumido: 2 itens

Itens no buffer: 3

Taxa de ocupação: 75%

- 27 Crie um programa em C++ que implemente uma fila de impressão circular com limite de documentos. O sistema gerencia documentos enviados por múltiplos usuários para uma impressora compartilhada. Quando a fila atinge a capacidade máxima, novos documentos substituem os mais antigos (política de substituição circular).

Cada documento tem: ID, nome do usuário, nome do arquivo e número de páginas. O sistema deve mostrar a ordem de impressão e quais documentos foram substituídos quando a fila estava cheia.

Este exercício aplica fila circular com política de substituição em um cenário real de gerenciamento de recursos compartilhados.

Exemplo de entrada

Capacidade da fila: 5 documentos

Usuário Ana envia: "relatorio.pdf" (10 páginas)

Usuário Bruno envia: "foto.jpg" (1 página)

Usuário Carla envia: "contrato.doc" (5 páginas)

Usuário Daniel envia: "apresentacao.ppt" (15 páginas)

Usuário Elisa envia: "planilha.xlsx" (3 páginas)

Fila cheia! (5/5 documentos)

Usuário Fabio envia: "artigo.pdf" (8 páginas)

Exemplo de saída

Fila de impressão:

1. Bruno - foto.jpg (1 pág)
2. Carla - contrato.doc (5 pág)
3. Daniel - apresentacao.ppt (15 pág)
4. Elisa - planilha.xlsx (3 pág)
5. Fabio - artigo.pdf (8 pág)

Documento substituído: Ana - relatorio.pdf (10 pág)

Total de páginas na fila: 32

Tempo estimado: 6 min 24 seg

EXERCÍCIOS DE ENADE

Ano	#	Curso	Conteúdo
2011	20	Ciência da Computação	Teoria de Grafos
2011	21	Ciência da Computação	Pilha e Fila no contexto da Genética
2011	24	Ciência da Computação	Fila de Prioridade (Heap)
2011	28	Ciência da Computação	Análise e Projeto de Algoritmos
2011	30	Ciência da Computação	Árvore Binária
2011	D03	Ciência da Computação	Fibonacci Recursivo
2011	D04	Ciência da Computação	Algoritmos de Ordenação e Listas Ordenadas
2014	D03	Ciência da Computação	Busca Recursiva
2014	D05	Ciência da Computação	Análise e Projeto de Algoritmos
2014	12	Ciência da Computação	Análise de Algoritmos
2014	13	Ciência da Computação	Árvore Binária
2014	16	Ciência da Computação	Pilha em C
2014	23	Ciência da Computação	Algoritmos Recursivos
2017	D03	Ciência da Computação	Lista Ligada
2017	D05	Ciência da Computação	Busca em Profundidade
2017	09	Ciência da Computação	Árvore Balanceada (AVL)
2017	18	Ciência da Computação	Algoritmos de Ordenação, Array em C
2017	22	Ciência da Computação	Algoritmo Guloso
2017	24	Ciência da Computação	Algoritmos em Grafos

Ano	#	Curso	Conteúdo
2017	25	Ciência da Computação	Análise de Fibonacci Recursivo
2017	33	Ciência da Computação	Função Recursiva
2019	D03	Engenharia da Computação	Árvore Binária e AVL
2019	D04	Engenharia da Computação	Produtório Iterativo vs Recursivo
2019	09	Engenharia da Computação	Algoritmo de Ordenação
2019	24	Engenharia da Computação	Spanning Tree no contexto de Algoritmos de Roteamento
2019	25	Engenharia da Computação	Busca Recursiva em Python
2021	D05	Ciência da Computação	Heap Binário
2021	20	Ciência da Computação	Análise de Algoritmos em C
2021	23	Ciência da Computação	Árvore Binária
2021	32	Ciência da Computação	Algoritmos de Ordenação
2021	34	Ciência da Computação	Algoritmo de Dijkstra
2023	D02	Engenharia da Computação	Algoritmo de Ordenação
2023	15	Engenharia da Computação	Vazamento de Memória
2023	18	Engenharia da Computação	Arrays Dinâmicos
2023	32	Engenharia da Computação	Busca Gulosa em Grafos
2023	34	Engenharia da Computação	Filas no contexto de Redes de Computadores

EXERCÍCIOS SABOR ENADE