Relatório Técnico: Implementação de Matrizes em C++

Equipe

9 de junho de 2025

1 Introdução

Este relatório descreve a implementação de uma biblioteca para manipulação de matrizes em C++, incluindo:

- Estruturas de dados utilizadas
- Divisão de módulos
- Descrição das rotinas
- Análise de complexidade (tempo e espaço)

2 Estruturas de Dados Utilizadas

2.1 Classe `Matriz`

Representa uma matriz genérica de dimensões $m \times n$.

Atributos:

- `linhas` (int): Número de linhas
- `colunas` (int): Número de colunas
- `dados` (`float**`): Matriz dinâmica alocada como um array de ponteiros

2.2 Classe `MatrizQuadrada` (Herda de `Matriz`)

Representa matrizes quadradas ($n \times n$), com operações específicas:

Métodos adicionais:

- `traco()`: Calcula o traço da matriz
- `determinante()`: Calcula o determinante (implementado apenas para 2×2)

3 Divisão de Módulos

O código está organizado em:

1. Definição da Classe `Matriz` (operações básicas e alocação dinâmica)
2. Classe `MatrizQuadrada` (extensão para matrizes quadradas)
3. Operações Matemáticas (sobrecarga de operadores)
4. Interface do Usuário (menu interativo no `main()`)

4 Descrição das Rotinas Principais

4.1 Operações Básicas (`Matriz`)

Método/Função	Descrição
`Matriz(int l, int c)`	Construtor que aloca memória para uma matriz $l \times c$
`~Matriz()`	Destrutor que libera a memória alocada
`preencher(string)`	Preenche a matriz a partir de uma string no formato `[[a,b],[c,d]]`
`alterar(i, j, valor)`	Modifica o valor na posição (i, j)
`imprimir()`	Exibe a matriz no console

4.2 Operações Matemáticas

Operação	Descrição
`A + B` (`operator+`)	Soma duas matrizes (retorna uma nova matriz ou erro se dimensões incompatíveis)
`A - B` (`operator-`)	Subtrai duas matrizes
`A * escalar`	Multiplica a matriz por um escalar
`A * B`	Multiplicação de matrizes (retorna `nullptr` se dimensões incompatíveis)
`transposicao(A)`	Retorna a matriz transposta

4.3 Métodos Específicos (`MatrizQuadrada`)

Método	Descrição
`traco()`	Soma os elementos da diagonal principal
`determinante()`	Calcula o determinante (apenas para matrizes 2×2)

5 Análise de Complexidade

5.1 Complexidade de Tempo

Operação	Complexidade	Explicação
`Matriz::Matriz(l, c)`	$O(l \times c)$	Alocação de memória para $l \times c$ elementos
`Matriz::preencher(str)`	$O(l \times c)$	Processa cada elemento da string e armazena na matriz
`A + B / A - B`	$O(n^2)$	Percorre todos os elementos das duas matrizes
`A * escalar`	$O(n^2)$	Percorre todos os elementos da matriz
`A * B`	$O(n^3)$	Triplo loop para multiplicação de matrizes
`transposicao(A)`	$O(n^2)$	Percorre todos os elementos e os reposiciona
`traco()`	$O(n)$	Percorre apenas a diagonal principal
`determinante()`	$O(1)$	Cálculo direto para matriz 2×2

5.2 Complexidade de Espaço

Operação	Complexidade	Explicação
`Matriz(l, c)`	$O(l \times c)$	Armazena uma matriz $l \times c$
`A + B / A - B`	$O(n^2)$	Gera uma nova matriz de mesmo tamanho
`A * B`	$O(n^2)$	Retorna uma matriz de tamanho lin_A $\times$ col_B
`transposicao(A)`	$O(n^2)$	Gera uma nova matriz transposta

6 Conclusão

O código implementa uma estrutura eficiente para manipulação de matrizes, com:

- Alocação dinâmica para evitar desperdício de memória
- Sobrecarga de operadores para facilitar operações matemáticas
- Tratamento de matrizes quadradas com métodos específicos

- Complexidade computacional otimizada para operações básicas