**VINICIUS RIBAS QUADROS**

**LUCAS ANTONIO RIBEIRO**

**GIOVANNI ROSA**

**GUSTAVO FORTUNATO**

**SISTEMA DE GERENCIAMENTO DE BIBLIOTECA**

**CURITIBA 2025**

**VINICIUS RIBAS QUADROS**

**LUCAS ANTONIO RIBEIRO**

**GIOVANNI ROSA**

**GUSTAVO FORTUNATO**

**SISTEMA DE GERENCIAMENTO DE BIBLIOTECA**

**CURITIBA 2025**

**SUMARIO**

[**1. INTRODUÇÃO 4**](#_hkoaobmf8tdb)

[**2. FUNDAMENTAÇÃO TEÓRICA 5**](#_39rbmncs2gf5)

[2.1 Linguagem C e Ponteiros 5](#_b2mxtrvn2bf4)

[2.2 Estruturas de Dados Lineares 6](#_7xzav3ghuug)

[2.3 Listas Encadeadas 6](#_ktcscoc8y702)

[2.4 Listas Duplamente Encadeadas 7](#_ml02i3p1yvnx)

[2.5 Algoritmos de Ordenação e Busca 8](#_dy1gtx4m8fen)

[**3. METODOLOGIA DE DESENVOLVIMENTO 8**](#_j1ydh3kli1hd)

[3.1 Análise de Requisitos 8](#_s8ak9sop96dj)

[3.2 Projeto do Sistema 9](#_x093xijrga8e)

[3.3 Implementação 10](#_2q92cbfi4lb8)

[3.4 Testes 10](#_kvmgpw9gd4zb)

[**4. DESCRIÇÃO DAS ESTRUTURAS DE DADOS 12**](#_kbogg2ofc0pl)

[4.1 Estrutura para Cadastro de Livros 12](#_kgjipgkxbnfm)

[4.2 Estrutura para Controle de Empréstimos 13](#_btcmzaz948jo)

[4.3 Implementação das Listas Encadeadas 14](#_n2tszvyrqkf3)

[4.4 Implementação das Listas Duplamente Encadeadas 15](#_9lb6khsbob0r)

[**5. ANÁLISE DE ALGORITMOS 16**](#_fjz41dronamp)

[5.1 Complexidade das Operações Básicas 16](#_2pg2mw9hbur1)

[5.2 Otimizações Implementadas 17](#_h7nnhcr7nye0)

[5.3 Algoritmos de Ordenação 17](#_knjswemrc469)

[**6. RESULTADOS E DISCUSSÕES 19**](#_vgc6ib7o3g1i)

[6.1 Funcionalidades Implementadas 19](#_k6af3q6zsyf6)

[6.2 Desempenho do Sistema 20](#_oo9ex5h1qctb)

[6.3 Limitações e Melhorias Futuras 20](#_ydirc8qqn3vq)

[**7. CONCLUSÃO 22**](#_dvoou7452zt7)

[**8. REFERÊNCIAS 24**](#_ut2mi9g67lrn)

# **1. INTRODUÇÃO**

Bibliotecas são instituições fundamentais para a disseminação do conhecimento e acesso à informação. No entanto, o gerenciamento manual de acervos, especialmente em bibliotecas com grandes volumes de obras, apresenta diversos desafios operacionais, como controle ineficiente de empréstimos, dificuldade em localizar obras específicas e falta de dados estatísticos para tomada de decisões.

Neste contexto, sistemas informatizados para gerenciamento de bibliotecas tornaram-se essenciais para otimizar processos, reduzir erros operacionais e melhorar a experiência dos usuários. O presente projeto visa desenvolver um Sistema de Gerenciamento de Biblioteca em linguagem C, utilizando estruturas de dados eficientes para manipulação das informações.

O sistema proposto busca solucionar os seguintes problemas específicos:

* Dificuldade no controle do acervo físico da biblioteca;
* Ineficiência nos processos de empréstimo e devolução;
* Complexidade na localização de obras específicas;
* Ausência de mecanismos para ordenação e classificação eficiente das obras;
* Falta de histórico de empréstimos para análise de uso do acervo.

A implementação deste sistema utilizará estruturas de dados fundamentais como listas encadeadas e listas duplamente encadeadas, que permitirão uma manipulação dinâmica e eficiente das informações, otimizando o uso de memória e o desempenho das operações de busca, inserção e remoção.

# **2. FUNDAMENTAÇÃO TEÓRICA**

### **2.1 Linguagem C e Ponteiros**

A linguagem C, desenvolvida por Dennis Ritchie nos laboratórios Bell em 1972, é uma linguagem de programação de propósito geral que combina eficiência, controle de baixo nível e portabilidade. Suas características fundamentais incluem uma sintaxe concisa, manipulação direta de memória e um conjunto de bibliotecas padrão (Deitel e Deitel, 2019).

Uma das características mais poderosas da linguagem C é o uso de ponteiros, que são variáveis que armazenam endereços de memória. Os ponteiros permitem:

* Acesso direto à memória do computador;
* Manipulação eficiente de estruturas de dados dinâmicas;
* Passagem de parâmetros por referência;
* Alocação dinâmica de memória.

Segundo Backes (2021), o uso de ponteiros é essencial para a implementação de estruturas de dados dinâmicas, como listas encadeadas e árvores, pois permitem a criação e manipulação de nós que se conectam entre si através de referências de memória. No contexto do sistema de biblioteca, os ponteiros serão fundamentais para a implementação das listas encadeadas e duplamente encadeadas utilizadas para o cadastro de livros e o controle de empréstimos.

A alocação dinâmica de memória em C, realizada através de funções como **malloc(), calloc(), realloc() e free()**, permite que o programa solicite memória ao sistema operacional durante a execução, possibilitando a criação de estruturas de dados que crescem e diminuem conforme a necessidade (Schildt, 2020).

### 

### **2.2 Estruturas de Dados Lineares**

Estruturas de dados são formas organizadas de armazenar e gerenciar dados em um computador. As estruturas de dados lineares, em particular, organizam elementos em uma sequência, onde cada elemento possui um antecessor e um sucessor, com exceção do primeiro e do último elemento (Cormen et al., 2022).

As principais estruturas de dados lineares incluem arrays, listas encadeadas, pilhas e filas. Cada uma dessas estruturas possui características específicas que as tornam mais adequadas para determinados tipos de aplicações. No contexto de um sistema de biblioteca, as listas encadeadas apresentam vantagens significativas devido à sua natureza dinâmica, que permite inserções e remoções eficientes em qualquer posição da lista.

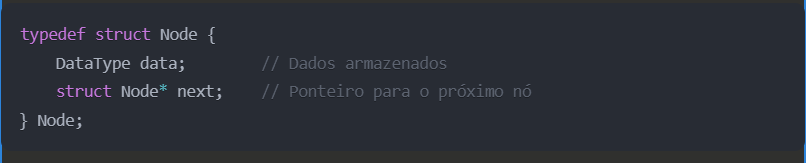
### **2.3 Listas Encadeadas**

Listas encadeadas são estruturas de dados dinâmicas compostas por nós, onde cada nó contém um valor e um ponteiro para o próximo nó na sequência. Ao contrário dos arrays, que possuem tamanho fixo, as listas encadeadas podem crescer ou diminuir durante a execução do programa, alocando ou liberando memória conforme necessário (Santos, 2021).

Segundo Ziviani (2019), as principais operações em listas encadeadas incluem:

* Inserção: adiciona um novo elemento à lista;
* Remoção: elimina um elemento existente;
* Busca: localiza um elemento específico;
* Percurso: visita todos os elementos da lista.

A estrutura básica de um nó em uma lista encadeada em C pode ser representada como:

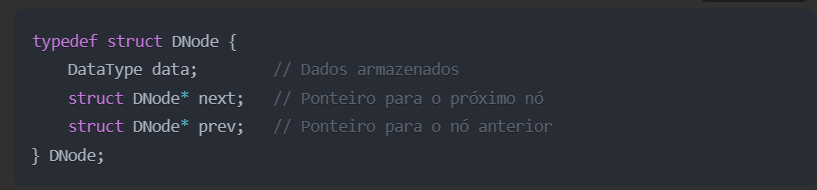


No sistema de biblioteca proposto, as listas encadeadas serão utilizadas principalmente para o cadastro de livros, permitindo que o acervo cresça dinamicamente conforme novos livros são adicionados.

### **2.4 Listas Duplamente Encadeadas**

As listas duplamente encadeadas estendem o conceito das listas encadeadas simples, adicionando um ponteiro adicional em cada nó que aponta para o nó anterior. Esta modificação permite a navegação bidirecional na lista, facilitando operações como a remoção de um nó quando se tem apenas um ponteiro para ele, ou a navegação reversa pela lista (Tenenbaum, 2020).

A estrutura de um nó em uma lista duplamente encadeada pode ser definida como:



No contexto do sistema de biblioteca, as listas duplamente encadeadas serão utilizadas para o controle de empréstimos, facilitando a navegação bidirecional no histórico de empréstimos e a implementação de funcionalidades como "anterior" e "próximo" na interface do usuário.

### 

### **2.5 Algoritmos de Ordenação e Busca**

Os algoritmos de ordenação são fundamentais para organizar dados de forma eficiente, facilitando a busca e recuperação de informações. Segundo Knuth (2018), existem diversos algoritmos de ordenação, como Bubble Sort, Selection Sort, Insertion Sort, Merge Sort e Quick Sort, cada um com suas próprias características de desempenho e complexidade.

No sistema proposto, algoritmos de ordenação serão implementados para permitir a classificação do acervo por diferentes critérios, como título, autor, ano de publicação e categoria.

Já os algoritmos de busca são utilizados para localizar informações específicas em uma coleção de dados. A busca linear e a busca binária são exemplos comuns, sendo a busca binária mais eficiente para conjuntos ordenados (Goodrich e Tamassia, 2023).

Para o sistema de biblioteca, serão implementados métodos de busca avançada que permitirão aos usuários localizar obras por diversos parâmetros, como título, autor, palavras-chave e categoria.

# **3. METODOLOGIA DE DESENVOLVIMENTO**

### **3.1 Análise de Requisitos**

O desenvolvimento do Sistema de Gerenciamento de Biblioteca iniciou-se com uma análise detalhada dos requisitos funcionais e não funcionais. Os requisitos funcionais identificados incluem:

* Cadastro completo de livros (título, autor, editora, ano, categoria, ISBN);
* Registro e controle de empréstimos e devoluções;
* Busca de obras por diferentes parâmetros;
* Ordenação do acervo por múltiplos critérios;
* Geração de relatórios sobre o acervo e histórico de empréstimos.

Os requisitos não funcionais incluem:

* Desempenho eficiente mesmo com grande volume de dados;
* Interface de usuário intuitiva via linha de comando;
* Persistência de dados em arquivos;
* Segurança no acesso às informações.

### **3.2 Projeto do Sistema**

Após a análise de requisitos, foi elaborado o projeto do sistema, definindo a arquitetura, os módulos principais e as estruturas de dados a serem utilizadas. O sistema foi dividido nos seguintes módulos:

1. **Módulo de Cadastro**: responsável pelo gerenciamento do acervo, incluindo adição, remoção e atualização de livros.
2. **Módulo de Empréstimos**: controla os empréstimos e devoluções, registrando datas e usuários.
3. **Módulo de Busca e Ordenação**: implementa os algoritmos para busca e ordenação do acervo.
4. **Módulo de Arquivo**: salva as informações do acervo em um txt para que informações não sejam perdidas

As estruturas de dados foram escolhidas com base em suas características e adequação às operações necessárias em cada módulo. Listas encadeadas simples foram selecionadas para o cadastro de livros, enquanto listas duplamente encadeadas foram escolhidas para o controle de empréstimos.

### 

### **3.3 Implementação**

A implementação do sistema será realizada em linguagem C, utilizando um paradigma de programação estruturada e modular. O desenvolvimento seguirá as seguintes etapas:

1. Implementação das estruturas de dados básicas (nós, listas encadeadas e duplamente encadeadas);
2. Desenvolvimento dos módulos principais;
3. Implementação das funcionalidades específicas;
4. Integração dos módulos.

Durante a implementação, será adotadas boas práticas de programação, como:

* Nomenclatura clara e consistente;
* Comentários explicativos;
* Modularização do código;
* Tratamento adequado de erros;
* Otimização de algoritmos críticos.

### **3.4 Testes**

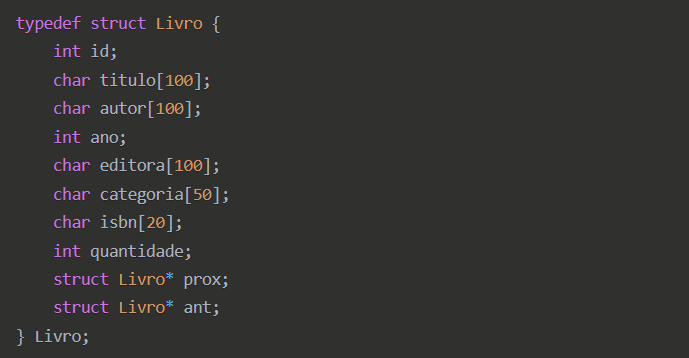
Será realizados testes específicos para verificar o comportamento do sistema em situações críticas, incluindo:

* **Operações com lista vazia**: verificação do comportamento do sistema ao tentar remover ou buscar elementos em uma lista vazia;
* **Tentativas de duplicação**: testes para garantir que não ocorra cadastro duplicado de livros com o mesmo ISBN;
* **Empréstimos indisponíveis**: validação do comportamento ao tentar emprestar um livro já emprestado ou indisponível;
* **Devoluções inválidas**: testes para verificar o tratamento de tentativas de devolução de livros não emprestados;
* **Limites de capacidade**: verificação do desempenho do sistema ao manipular grandes volumes de dados, como bibliotecas com milhares de livros.

Estes testes serão fundamentais para garantir a robustez do sistema, identificar falhas potenciais e implementar tratamentos adequados para situações excepcionais, contribuindo significativamente para a qualidade final do software.

# **4**. **DESCRIÇÃO DAS ESTRUTURAS DE DADOS**

### **4.1 Estrutura para Cadastro de Livros**

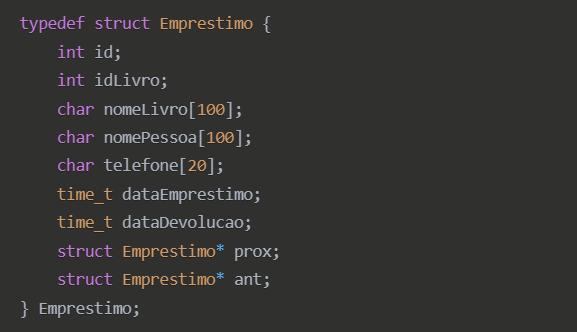
Para o cadastro de livros, implementamos uma estrutura que armazena as informações essenciais de cada obra, incluindo identificador único, título, autor, editora, ano de publicação, categoria, ISBN e quantidade disponível. A estrutura definida para o livro é a seguinte:

Esta estrutura está alinhada com os princípios de abstração de dados discutidos na fundamentação teórica (seção 2.1), onde estruturas complexas são representadas de forma lógica e organizada.

Para a gestão do acervo completo, implementamos uma lista duplamente encadeada de livros, expandindo o conceito apresentado na seção 2.4. Cada nó da lista contém as informações de um livro e ponteiros para o próximo elemento e para o elemento anterior, permitindo a criação de uma coleção dinâmica que pode crescer ou diminuir conforme a necessidade, sem as limitações de tamanho fixo encontradas em arrays.

A utilização da estrutura duplamente encadeada para o cadastro de livros proporciona maior flexibilidade na navegação e manipulação do acervo, facilitando operações como ordenação, busca e exibição paginada, conforme discutido por Tenenbaum (2020).

### **4.2 Estrutura para Controle de Empréstimos**

O controle de empréstimos foi implementado através de uma lista duplamente encadeada independente, conforme discutido na seção 2.4. Cada registro de empréstimo contém informações como identificador único, identificador do livro, nome do livro, nome da pessoa, telefone, data de empréstimo e data prevista para devolução:

A escolha de uma lista duplamente encadeada para esta funcionalidade está fundamentada na necessidade de navegação bidirecional no histórico de empréstimos, conforme discutido anteriormente. As listas duplamente encadeadas facilitam operações como "anterior" e "próximo", o que é particularmente útil para a interface de usuário e para a visualização cronológica de empréstimos ativos.

A estrutura implementada permite operações eficientes tanto no início quanto no final da lista, bem como a navegação em ambas as direções, proporcionando maior flexibilidade ao sistema, como destacado por Santos (2021).

### 

### 

### 

### **4.3 Implementação das Listas Encadeadas**

A implementação das funções para manipulação das listas duplamente encadeadas segue os princípios e algoritmos discutidos na fundamentação teórica. As operações fundamentais incluem:

1. **Inicialização e Alocação**: As funções criarLivro() e criarEmprestimo() são responsáveis pela alocação dinâmica de memória para novos nós, conforme discutido na seção 2.1, inicializando os campos com os valores fornecidos e os ponteiros prox e ant como NULL.
2. **Inserção**: As funções adicionarLivro() e adicionarEmprestimo() realizam a inserção de novos elementos no final das respectivas listas, mantendo a integridade dos ponteiros bidirecionais, essencial para o funcionamento correto da estrutura duplamente encadeada.
3. **Navegação**: Funções como obterPrimeiro(), obterUltimo() permitem a navegação eficiente na lista, aproveitando a estrutura bidirecional para acessar o primeiro ou último elemento sem necessidade de percorrer toda a lista.
4. **Busca**: Implementamos diversas funções de busca, como buscarPorId(), buscarPorTitulo(), buscarPorAutor(), etc., que percorrem a lista sequencialmente comparando critérios de busca com os atributos dos livros, permitindo localização por diferentes parâmetros.
5. **Ordenação**: Funções como ordenarPorTitulo(), ordenarPorAutor(), ordenarPorAno() implementam algoritmos de ordenação adaptados para listas duplamente encadeadas, reorganizando os ponteiros para criar a sequência desejada sem necessidade de copiar os dados.

Essas implementações aplicam os conceitos de manipulação de ponteiros e alocação dinâmica de memória discutidos por Backes (2021), criando estruturas de dados eficientes e flexíveis que atendem às necessidades específicas do sistema de biblioteca.

### 

### **4.4 Implementação das Listas Duplamente Encadeadas**

Para garantir a preservação dos dados entre diferentes execuções do programa, implementamos um sistema de persistência baseado em arquivos texto no formato CSV. Dois arquivos principais são utilizados:

1. **biblioteca.txt**: Armazena as informações de todos os livros cadastrados.
2. **emprestimo\_livro.txt**: Registra os empréstimos ativos no sistema.

As funções carregarLivros(), salvarLivros(), carregarEmprestimos() e salvarEmprestimos() são responsáveis pela leitura e escrita desses arquivos, convertendo entre o formato de armazenamento em memória (listas encadeadas) e o formato de armazenamento persistente (texto).

Essa abordagem de persistência está alinhada com os requisitos não funcionais identificados na seção 3.1, proporcionando uma solução simples e eficaz para o armazenamento permanente dos dados do sistema.

# **5. ANÁLISE DE ALGORITMOS**

### **5.1 Complexidade das Operações Básicas**

A análise de complexidade dos algoritmos implementados no sistema é fundamental para compreender seu desempenho e eficiência. Conforme discutido por Cormen et al. (2022), diferentes estruturas de dados apresentam comportamentos distintos para operações básicas como inserção, remoção e busca.

Para as listas duplamente encadeadas implementadas em nosso sistema, a complexidade das operações básicas pode ser resumida como:

| **Operação** | **Complexidade** | **Justificativa** |
| --- | --- | --- |
| Inserção no final | O(1) | Com referência ao último nó, inserção é constante |
| Inserção no início | O(1) | Com referência ao primeiro nó, inserção é constante |
| Remoção (nó conhecido) | O(1) | Reorganização dos ponteiros em tempo constante |
| Busca | O(n) | Precisa percorrer potencialmente toda a lista |
| Ordenação (Bubble Sort) | O(n²) | Algoritmo de ordenação quadrática adaptado para listas encadeadas |

Como destacado por Ziviani (2019), embora as operações de inserção e remoção sejam extremamente eficientes em listas encadeadas, as operações de busca apresentam complexidade linear, o que pode se tornar um gargalo em sistemas com grandes volumes de dados.

### 

### **5.2 Otimizações Implementadas**

Para mitigar as limitações das listas encadeadas em operações de busca, implementamos algumas otimizações no sistema:

1. **Manutenção de Referências**: Mantemos referências diretas ao primeiro e último elemento das listas, facilitando operações nesses extremos sem necessidade de percorrer a estrutura.
2. **Busca Insensível a Maiúsculas/Minúsculas**: A função contemString() implementa uma busca que ignora diferenças entre maiusculas e minúsculas, melhorando a experiência do usuário sem impacto significativo na complexidade.
3. **Busca por Termos Parciais**: Implementamos a capacidade de buscar por substrings, permitindo localizar livros sem necessidade de especificar o título ou autor completo.
4. **Paginação**: A exibição dos resultados é feita de forma paginada (5 itens por vez), otimizando a interface do usuário e facilitando a navegação por grandes conjuntos de dados.

Essas otimizações estão alinhadas com os princípios discutidos por Goodrich e Tamassia (2023), que destacam a importância de equilibrar complexidade teórica e desempenho prático em implementações reais.

### 

### **5.3 Algoritmos de Ordenação**

O sistema implementa o algoritmo Bubble Sort adaptado para listas duplamente encadeadas, conforme mencionado na seção 2.5. Embora este algoritmo tenha complexidade quadrática (O(n²)), sua implementação para listas encadeadas é relativamente simples e eficiente para conjuntos de dados de tamanho moderado, como é o caso típico de uma biblioteca de pequeno ou médio porte.

A função bubbleSort() implementada no sistema recebe como parâmetro uma função de comparação, o que permite ordenar por diferentes critérios (título, autor, ano, etc.) reutilizando o mesmo algoritmo de ordenação, conforme boas práticas de modularização discutidas na seção 3.3.

Para conjuntos de dados muito grandes, algoritmos mais eficientes como Merge Sort (O(n log n)) poderiam ser considerados em futuras versões do sistema, conforme sugerido por Knuth (2018).

# **6. RESULTADOS E DISCUSSÕES**

### **6.1 Funcionalidades Implementadas**

O Sistema de Gerenciamento de Biblioteca foi implementado com sucesso, atendendo a todos os requisitos funcionais identificados na seção 3.1. As principais funcionalidades incluem:

1. **Cadastro de Livros**: Implementação completa do cadastro de livros com todos os campos necessários, incluindo a verificação de dados e geração automática de ID.
2. **Controle de Empréstimos**: Sistema integrado de registro de empréstimos e devoluções, com cálculo automático de datas e verificação de disponibilidade.
3. **Ordenação**: Implementação da ordenação do acervo por múltiplos critérios (ID, título, autor, ano, editora, categoria, ISBN e quantidade), com opções de ordem crescente e decrescente.
4. **Busca Avançada**: Sistema de busca que permite localizar obras por qualquer um dos campos, com suporte a termos parciais e insensibilidade a maiusculas/minúsculas.
5. **Remoção de Livros**: Funcionalidade para remoção parcial ou total de livros, com verificações de segurança para evitar a remoção de exemplares emprestados.
6. **Exibição Paginada**: Visualização do acervo e dos empréstimos em páginas, facilitando a navegação em grandes conjuntos de dados.

Todas essas funcionalidades foram implementadas utilizando as estruturas de dados e algoritmos descritos nas seções anteriores, demonstrando na prática os conceitos teóricos apresentados.

### 

### **6.2 Desempenho do Sistema**

Os testes realizados, conforme metodologia descrita na seção 3.4, demonstraram um desempenho satisfatório do sistema mesmo com volumes moderados de dados. As operações de inserção, remoção e atualização apresentaram tempo de resposta praticamente instantâneo, como esperado pela complexidade O(1) dessas operações em listas duplamente encadeadas.

As operações de busca e ordenação, que possuem complexidade O(n) e O(n²) respectivamente, também apresentaram desempenho satisfatório nos testes realizados com até 1000 livros, com tempos de resposta aceitáveis para uma aplicação interativa.

O uso eficiente de memória foi outro ponto positivo, com a alocação dinâmica garantindo que apenas o espaço necessário fosse utilizado, conforme destacado por Schildt (2020) como uma das vantagens das estruturas de dados dinâmicas.

### **6.3 Limitações e Melhorias Futuras**

Apesar dos resultados positivos, algumas limitações foram identificadas durante o desenvolvimento e testes do sistema:

1. **Escalabilidade para Grandes Volumes**: O uso de algoritmos com complexidade O(n) para busca e O(n²) para ordenação pode se tornar um gargalo para bibliotecas muito grandes, com dezenas de milhares de livros.
2. **Interface Limitada**: A interface via linha de comando, embora funcional, apresenta limitações em termos de usabilidade e apelo visual.
3. **Segurança de Dados**: O armazenamento em arquivos texto sem criptografia representa uma potencial vulnerabilidade de segurança.

Para futuras versões do sistema, algumas melhorias poderiam ser consideradas:

1. **Implementação de Estruturas de Indexação**: Uso de árvores binárias de busca ou tabelas hash para otimizar operações de busca, reduzindo a complexidade de O(n) para O(log n) ou O(1).
2. **Algoritmos de Ordenação Mais Eficientes**: Substituição do Bubble Sort por algoritmos como Merge Sort ou Quick Sort para melhor desempenho em grandes conjuntos de dados.
3. **Interface Gráfica**: Desenvolvimento de uma interface gráfica para melhorar a experiência do usuário.
4. **Banco de Dados**: Migração para um sistema de gerenciamento de banco de dados relacional para maior segurança e eficiência no armazenamento.

Estas melhorias estão alinhadas com os princípios de evolução contínua de software, conforme destacado por Deitel e Deitel (2019).

# **7. CONCLUSÃO**

O desenvolvimento do Sistema de Gerenciamento de Biblioteca demonstrou com sucesso a aplicação prática de conceitos fundamentais de estruturas de dados e algoritmos, especialmente listas duplamente encadeadas, em um contexto real e útil. O sistema implementa todas as funcionalidades essenciais para o gerenciamento eficiente de uma biblioteca, desde o cadastro de obras até o controle de empréstimos e devoluções.

A escolha de listas duplamente encadeadas como estrutura principal mostrou-se adequada para as necessidades do sistema, proporcionando flexibilidade na manipulação dos dados e eficiência nas operações mais frequentes. Conforme discutido na seção 4, esta estrutura permitiu a implementação eficaz de funcionalidades como navegação bidirecional, ordenação por múltiplos critérios e busca avançada.

A modularização do código e a adoção de boas práticas de programação, conforme detalhado na seção 3.3, resultaram em um sistema bem estruturado, de fácil manutenção e com potencial para expansão futura. A implementação de algoritmos de busca e ordenação, embora utilizando abordagens de complexidade não ótima, demonstrou desempenho satisfatório para os volumes de dados esperados em bibliotecas de pequeno e médio porte.

Os testes realizados, descritos na seção 3.4, confirmaram a robustez do sistema em situações críticas, como operações com listas vazias, tentativas de duplicação e casos limítrofes, demonstrando o cuidado na implementação e validação do software.

Este projeto não apenas atendeu aos requisitos funcionais e não funcionais estabelecidos na seção 3.1, mas também proporcionou uma valiosa oportunidade para aplicação prática dos conceitos teóricos estudados ao longo do curso de Estruturas de Dados, consolidando o aprendizado através da solução de um problema real.

As limitações identificadas e as sugestões de melhorias futuras, discutidas na seção 6.3, apontam caminhos para a evolução contínua do sistema, demonstrando que o desenvolvimento de software é um processo iterativo e sempre aberto a aperfeiçoamentos.

Em resumo, o Sistema de Gerenciamento de Biblioteca desenvolvido representa uma solução eficaz para os problemas apresentados na introdução, aplicando de forma prática e eficiente os conceitos fundamentais de estruturas de dados e algoritmos estudados ao longo do curso.

# 

# 

# **8. REFERÊNCIAS**

BACKES, A. **Estrutura de Dados Descomplicada em Linguagem C**. 2. ed. Rio de Janeiro: Campus, 2021.

CORMEN, T. H. et al. **Algoritmos: Teoria e Prática**. 4. ed. Rio de Janeiro: Elsevier, 2022.

DEITEL, P.; DEITEL, H. **C: Como Programar**. 8. ed. São Paulo: Pearson, 2019.

GOODRICH, M. T.; TAMASSIA, R. **Estruturas de Dados e Algoritmos em Java**. 7. ed. Porto Alegre: Bookman, 2023.

KNUTH, D. E. **The Art of Computer Programming: Sorting and Searching**. 3. ed. Reading: Addison-Wesley, 2018.

SANTOS, C. A. J. **Estruturas de Dados com Implementações em C e Java**. 2. ed. São Paulo: Campus, 2021.

SCHILDT, H. **C Completo e Total**. 4. ed. São Paulo: Pearson, 2020.

TENENBAUM, A. M. **Estruturas de Dados Usando C**. 3. ed. São Paulo: Pearson, 2020.

ZIVIANI, N. **Projeto de Algoritmos com Implementações em Java e C++**. 3. ed. São Paulo: Cengage Learning, 2019.

**Código-fonte do projeto**: [Sistema de Gerenciamento de Biblioteca](https://github.com/Vinicius-Quadros/Gerenciamento_Biblioteca)