RELATÓRIO TÉCNICO SOBRE O DESEMPENHO DE ÁRVORES RUBRO NEGRAS E ÁRVORES 2-3

Sthefany Moura Godinho

RESUMO

Este relatório apresenta a implementação e a avaliação, em `C`, de um índice de artistas baseado em duas árvores balanceadas rubro–negra (RB) e Árovre 2–3 empregadas no módulo de biblioteca musical. O foco experimental é a busca por nome. As árvores são previamente carregadas a partir do arquivo sintético `biblioteca_30.txt`. O lote de consultas é fixo (`QUANT`=30), formado por 24 nomes reais coletados por percurso em ordem e 6 nomes inexistentes gerados automaticamente. Para evitar realocações durante a medição, os nomes são armazenados em *buffer* fixo de 100 caracteres. Os tempos são aferidos em Linux (Ubuntu 24.04.3 LTS) com clock() (ms), sem *I/O* dentro do laço. O caminho percorrido é impresso *após* o cronômetro. Resultados: Árvore 2–3 obteve 24/30 encontrados, 0,053 ms de tempo total e 0,002 ms por busca (média exibida), com 5,00 nós visitados em média; RB obteve 24/30, 0,057 ms no total, 0,002 ms por busca e 4,40 nós visitados. Como ambas têm custo $O(\log n)$, as diferenças observadas são pequenas e atribuíveis a constantes de implementação, ainda assim, nesta instância a árvore 2–3 apresentou *tempo total* ligeiramente menor, enquanto a RB percorreu menos nós em média. Conclui-se que as duas estruturas entregam buscas previsíveis. A escolha prática pode considerar simplicidade de implementação, padrões de acesso e requisitos de manutenção. Recomenda-se ampliar a amostra (mais artistas/consultas) e utilizar cronômetros de maior resolução para análises mais robustas.

1 INTRODUÇÃO

O crescimento contínuo do acervo de conteúdos digitais exige estruturas de dados capazes de armazenar e recuperar informações com eficiência. No contexto deste trabalho, desenvolve-se, em linguagem C, uma Biblioteca de Música organizada hierarquicamente em Artistas → Álbuns → Músicas, com integridade referencial garantida entre os níveis.

Na camada de dados, comparamos duas árvores balanceadas para indexar *Artistas* por nome: a Árvore Rubro–Negra (RB) e a Árvore 2–3. Os *Álbuns* utilizam a estrutura compatível com cada variante, e as *Músicas* permanecem em lista encadeada ordenada. A interface por menus oferece inserção, busca e remoção em todos os níveis, respeitando restrições de consistência (remoções em cascata quando aplicável).

A avaliação empírica concentra-se na busca por nome de artista. As árvores são previamente carregadas a partir do arquivo sintético `biblioteca_30.txt`; o lote de consultas é controlado (`QUANT=30`), combinando nomes existentes e inexistentes. As medições de tempo usam `clock()` (convertido para milissegundos) *sem I/O* dentro do intervalo cronometrado; o caminho percorrido na árvore é registrado e impresso *após* a medição.

Este relatório descreve a modelagem das estruturas, os principais módulos (inserção, busca, remoção e persistência), o desenho do experimento e os critérios de comparação (*tempo*

total/médio e *nós visitados*), preparando o terreno para a discussão dos resultados nas seções seguintes.

2 SEÇÕES ESPECÍFICAS

2.1 Questão 1 (Árvore Rubro–Negra)

A Questão 1 implementa uma Biblioteca de Música em C cuja camada de dados utiliza Árvore Rubro–Negra (RB-Tree) para organizar Artistas por *nome* (chave). Para cada artista armazenam-se *nome*, *estilo*, *tipo* (solo, dupla, banda etc.), *número de álbuns* e um ponteiro para a sua coleção de Álbuns. Cada álbum contém *título*, *ano*, *quantidade de músicas* e um ponteiro para a lista encadeada ordenada de Músicas. Cada música registra *título* e *duração (min)*.

A interface por menus permite *inserção*, *busca* e *remoção* em todos os níveis, respeitando as restrições de integridade: um álbum só é inserido em *artista já cadastrado* e uma música só é inserida em *álbum já cadastrado*. Há buscas locais (dentro do artista/álbum) e buscas globais (por álbum em toda a biblioteca; por música em todos os álbuns de todos os artistas). A remoção em cascata é informada ao usuário (remover artista ⇒ remove todos os álbuns e músicas; remover álbum ⇒ remove suas músicas). Em termos algorítmicos, a RB-Tree mantém o conjunto de artistas (e, opcionalmente, álbuns) *balanceado*, oferecendo buscas e inserções em $O(\log n)$ com boa estabilidade.

2.2 Questão 2 (Árvore 2–3)

A Questão 2 *replica a mesma funcionalidade* da Questão 1, porém substitui a estrutura de Artistas por uma Árvore 2–3. A organização hierárquica, os menus, as validações e a persistência permanecem idênticos. A Árvore 2–3 mantém nós com uma ou duas chaves, garantindo altura $O(\log n)$ e, portanto, custos de busca/inserção comparáveis à RB-Tree, porém com lógica distinta (quebras de nós, promoção de chaves, três filhos). Os Álbuns também são tratados com árvore (2–3) compatível com a versão; as Músicas seguem em *lista encadeada ordenada*. Essa variação permite comparar empiricamente duas estruturas balanceadas aplicadas ao mesmo domínio.

2.3 Requisitos Funcionais e Implementação

O sistema está em conformidade com a especificação da *Biblioteca de Música*, preservando a hierarquia Artistas → Álbuns → Músicas e a integridade referencial entre esses níveis. A seguir, apresentamos uma síntese dos módulos e das operações implementadas.

2.3.1 Artistas

Os artistas são indexados por *nome*. Na versão da Questão 1, a árvore é Rubro–Negra; na Questão 2, é árvore 2–3. Em ambos os casos:

- `alocaArtista`/`preencherArtista`: criação e entrada de dados do artista.
- `inserirArtista`: insere sem duplicar (balanceado por RB ou 2–3).
- `buscarArtista`/`buscarInfoArtista`: busca por nome (igualdade/ordenação lexicográfica).

- `imprimirArv`: impressão hierárquica da árvore de artistas.
- `removeArtista` (quando aplicável): remoção com liberação dos álbuns e suas músicas.
- `liberarArv`: desalocação total (pos-ordem), liberando subárvores e coleções associadas.

2.3.2 Álbuns

Cada artista mantém uma coleção de álbuns em árvore compatível com a variante (RB na Questão 1; árvore 2–3 na Questão 2), ordenada por *título*.

- `criarNoAlbum`/`preencherAlbum`: criação/entrada de dados do álbum.
- `inserirNoAlbum`: insere sem duplicar (aplicando regras de RB ou 2–3).
- `buscarAlbum`: busca por título (local ao artista ou global, percorrendo todos os artistas).
- `imprimirArvAlbum`: impressão da árvore de álbuns do artista.
- `removeAlbum`: remoção com liberação da lista de músicas.
- `liberarArvAlbum`: desalocação total dos nós e suas listas de músicas.

2.3.3 Músicas

As músicas de um álbum são mantidas em lista encadeada ordenada por *título*, sem duplicação.

- `alocarNo`/`preencherNo`: criação/entrada de dados da música.
- `inserirMusica`: insere na posição ordenada (ignora duplicata).
- `buscarMusica`: busca sequencial por título (local ao álbum; *global* percorre todos os álbuns).
- `mostrarMusicas`: listagem ordenada.
- `removerMusica`: remoção por título (ajuste de ponteiros).
- `liberarListaMusicas`: desalocação completa da lista.

2.3.4 Persistência (carregar/salvar)

O sistema realiza persistência em arquivo texto. Ao iniciar, `carregarBiblioteca` lê o arquivo e reconstrói a árvore de artistas, os álbuns de cada artista e as listas de músicas; ao encerrar, `salvarBiblioteca` grava o estado atual no mesmo formato. O carregamento respeita a hierarquia (`ARTISTA` → `ALBUM` → `MUSICA`) e atualiza contadores (p. ex., número de álbuns) ao inserir.

2.3.5 Menus e Fluxo de Uso

Há três menus hierárquicos: *Artistas* (nível raiz), *Álbuns* (de um artista) e *Músicas* (de um álbum). Cada menu oferece inserção, listagem, busca e remoção, além de buscas globais (por álbum e por música) que percorrem toda a biblioteca. Mensagens orientam o usuário sobre efeitos em cascata nas remoções.

2.3.6 Experimento de 30 buscas

Conforme o enunciado, foi implementado um experimento de 30 buscas por artistas, que:

1. Coleta até 24 nomes reais (ordem em-ordem) e completa com nomes inexistentes até totalizar 30.
2. Para cada busca, *cronometragem limpa* com `clock()`, *sem I/O* dentro do intervalo medido.
3. Registra o *caminho de nós visitados* e o imprime *após* a medição.
4. Reporta, para cada busca, *achou/não achou*, *nós visitados* e *tempo (ms)*; ao final, apresenta *resumo* com tempos total/médio e média de nós visitados.

O mesmo procedimento é aplicado às duas variantes (Rubro–Negra e 2–3), permitindo comparação direta.

Tabela 1 – Representação simplificada das estruturas principais

Estrutura	**Componentes principais**
`Artista`	infoArtista, esq/cen/dir (2–3) *ou* esq/dir+cor (RB), **albuns** → `Album`
`Album`	infoAlbum, esq/cen/dir (2–3) *ou* esq/dir+cor (RB), **musica** → lista
`Musica`	infoMusica, `proximo` (lista encadeada ordenada)

2.4 Tecnologias e Recursos Utilizados

Para melhor entendimento deste trabalho, é necessário conhecer e ter o entendimento das estruturas de dados utilizadas nos experimentos. Esta seção tem o intuito de introduzir os conceitos e características essenciais das estruturas usadas, além de apresentar os problemas utilizados nos testes, bem como as implementações das soluções para eles.

Componente	**Especificação**
Sistema Operacional	Ubuntu 24.04.1 LTS (Kernel 6.14.0-29-generic, x86$_6$4)
Processador	Intel Core i7-1165G7, 2.80GHz, 8 núcleos lógicos, 4 físicos
Memória RAM	7,4 GiB
Armazenamento	256 GB SSD
Compilador Copyright (C) 2023	gcc (Ubuntu) 13.3.0
Linguagem	C (com uso de ponteiros, structs e árvores binárias)
Medição de Tempo	`clock()` e `CLOCKS_PER_SEC` (`time.h`)

Tabela 2 – Especificações da Máquina e Ferramentas de Testes

3 RESULTADOS E DISCUSSÕES

3.1 Contexto dos Testes

Os experimentos comparam duas estruturas para indexar artistas por nome: **árvore 2–3** e **árvore rubro–negra (RB)**. Em ambas, a árvore já está carregada com os artistas do arquivo `biblioteca_30.txt`. Cada busca mede tempo com `clock()` (convertido para ms) sem `printf` dentro do laço (*tempo limpo*), registra o caminho (nós visitados) e imprime o caminho *após* a medição.

3.2 Metodologia

O lote de consultas tem tamanho fixo `QUANT = 30`. Para compor esse lote: (i) percorremos a árvore em-ordem para coletar nomes **reais** (até um limite `L`); (ii) completamos com nomes **inexistentes** gerados a partir de uma base (`<nome>__nao_existe_<id>`) até totalizar 30 entradas. Os nomes são armazenados em buffer fixo de 100 caracteres (com truncamento seguro). Nesta execução:

$$L_{2\text{–}3} = 24 \quad \text{e} \quad L_{\text{RB}} = 24,$$

resultando em 24 consultas reais + 6 sintéticas em ambas as estruturas.

3.2.0.1 Observação.

O número de encontrados coincide com o número de nomes reais do lote: **24/30** em ambas as estruturas, pois os nomes sintéticos são, por construção, ausentes.

3.3 Resultados

A Tabela 3 resume os indicadores emitidos pelo programa (`Resumo (2-3)` e `Resumo (RB)`) para `QUANT = 30`.

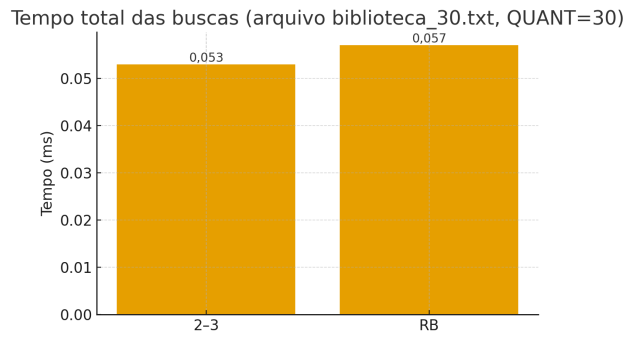
Tabela 3 – Resumo das buscas por nome de artista (arquivo `biblioteca_30.txt`)

Estrutura	Encontrados / 30	Tempo total (ms)	Tempo médio (ms)	Nós visitados (méd
RB	24/30	0.057	0.002	4.40
2–3	24/30	0.053	0.002	5.00

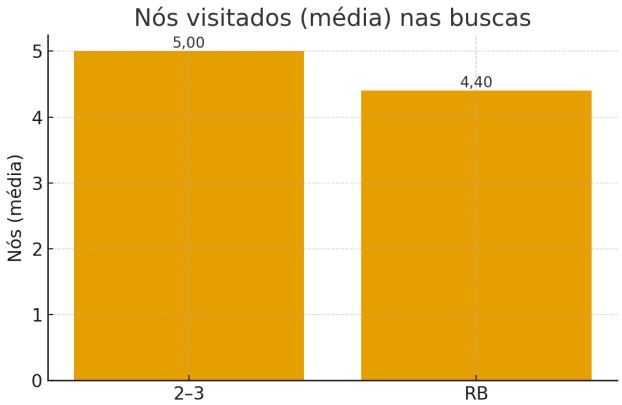
3.4 Discussão dos resultados

Os tempos absolutos são muito baixos (ordem de $\sim 10^{-3}$ ms por busca), portanto diferenças pequenas podem estar próximas da resolução de `clock()` e do ruído do ambiente. Ainda assim, vemos tendências condizentes com a estrutura:

- **Nós visitados.** Na 2–3, os caminhos ficaram praticamente constantes nesta instância (média **5.00**). Na RB, houve variação maior (1 a 6 nós nos exemplos), com média menor (**4.40**). Isso indica custo de comparação/navegação um pouco diferente entre as estruturas.
- **Tempo médio.** Muito semelhante nas duas (**0.002 ms** por busca na precisão exibida); a impressão do caminho ocorre fora da janela cronometrada e não afeta as medidas.



(a) Tempo total das buscas (`biblioteca_30.txt`, `QUANT=30`).



(b) Nós visitados (média) nas buscas.

Figura 1 – Comparativo de desempenho (RB vs. 2–3) em busca por nome.

- **Composição do lote.** Como agora $L_{2\text{–}3} = L_{\text{RB}} = 24$, o número de encontrados é o mesmo (24). Em cenários com mais ausentes, a média de nós visitados e o tempo total tendem a subir levemente.

3.4.0.1 Síntese.

Com `QUANT = 30` e este conjunto, a **2–3** obteve *tempo total* ligeiramente menor que a RB (0,053 ms vs. 0,057 ms), embora visite mais nós em média (5,00 vs. 4,40). Como ambos são balanceados e as diferenças são pequenas, recomenda-se repetir o experimento com cargas maiores para avaliar a estabilidade das médias.

4 CONCLUSÃO

O experimento teve como propósito avaliar, no índice de artistas, o efeito do *balanceamento* na **busca por nome**, comparando **árvore rubro–negra (RB)** e **árvore 2–3** sob condições controladas (dados sintéticos, lote fixo e cronometria limpa com `clock()`). O objetivo foi alcançado. Na execução com `biblioteca_30.txt` (`QUANT`=30, $L = 24$), ambas retornaram **24/30** encontrados; a **2–3** apresentou *tempo total* ligeiramente menor (0,053 ms) que a **RB** (0,057 ms), com *tempo médio* por busca igual na precisão exibida (0,002 ms). Descobriu-se que as diferenças são *marginais* e explicadas por *constantes de implementação* (nó multichave, padrões de ramificação e comparação *case-insensitive*), não por complexidade assintótica; a RB percorreu menos nós em média (4,40 vs. 5,00), mas isso não se traduziu em vantagem de tempo neste conjunto. Aprendeu-se, portanto, que ambas são adequadas para consultas pontuais e previsíveis; a escolha prática deve considerar simplicidade de implementação, padrões de acesso e manutenção. Como passo seguinte, recomenda-se ampliar amostras e consultas e empregar cronômetros de maior resolução para consolidar as tendências observadas.