UNIVERSIDADE FEDERAL DO PIAUÍ
CAMPUS SENADOR HELVÍDIO NUNES DE BARROS
CURSO DE SISTEMAS DE INFORMAÇÃO
DISCIPLINA DE ESTRUTURA DE DADOS II
PROFESSORA: JULIANA OLIVEIRA DE CARVALHO
ALUNO: VINICIUS DA SILVA NUNES

RELATÓRIO DE ESTRUTURA DE DADOS II

Resumo:

O relatório discute as estruturas de dados conhecidas como árvores rubro negras, árvores 2-3 e árvores 4-5, todas projetadas para armazenar informações de maneira eficiente, garantindo operações de busca, inserção e remoção com complexidade balanceada. Estas são consideradas árvores de busca binária balanceadas, criadas para manter o equilíbrio da árvore durante as operações de inserção e remoção de elementos.

A motivação para este relatório surgiu ao questionar como essas três estruturas de árvores se diferenciam em desempenho. O problema principal abordado envolve a implementação das árvores rubro-negras, 2-3 e 4-5 na linguagem C. O objetivo é criar funções que permitam a inserção, busca e remoção de elementos nessas árvores de maneira eficiente, preservando a propriedade de balanceamento.

Ao realizar a implementação das árvores rubro-negras, 2-3 e 4-5, observou-se que ambas são capazes de lidar eficientemente com grandes volumes de dados, mantendo a altura da árvore próxima ao logaritmo da quantidade de elementos. As operações de busca, inserção e remoção foram otimizadas, resultando em tempos de execução consistentes mesmo para conjuntos expressivos de dados.

Palavras-chaves: rubro-negra; 2-3; 4-5; estruturas de dados.

1 Introdução

As estruturas de dados desempenham um papel fundamental na ciência da computação, permitindo a organização eficiente, armazenamento e manipulação de informações. No entanto, a verdadeira utilidade dessas estruturas se manifesta quando são combinadas com algoritmos específicos que visam manipulá-las. Este relatório se concentra em três tipos de árvores de busca balanceadas: árvores rubro-negras, árvores 2-3 e árvores 4-5.

Árvores, como estruturas de dados, estabelecem relações hierárquicas entre os dados que contêm. Cada nó em uma árvore está conectado a um nó superior, chamado pai, e pode ter um ou mais nós abaixo dele, chamados filhos. A raiz da árvore é o nó sem pai, e os nós são organizados em níveis, formando subárvores. Essa hierarquia proporciona uma organização eficiente dos dados, facilitando operações como busca, inserção e remoção.

As árvores binárias rubro-negras, árvores 2-3 e árvores 4-5 são exemplos de árvores balanceadas cruciais para resolver problemas computacionais que demandam operações eficientes em estruturas de dados. Elas asseguram um desempenho logarítmico para operações fundamentais, sendo ideais para gerenciar grandes conjuntos de dados. Cada tipo de árvore apresenta características específicas que contribuem para o equilíbrio da estrutura e, por conseguinte, para um desempenho eficaz das operações. O objetivo deste relatório é examinar e comparar o desempenho de três tipos de estruturas de dados - rubro-negras, 2-3 e 4-5 - e analisar as estratégias de busca, inserção e remoção utilizadas. Além disso, serão discutidas as propriedades e vantagens de cada tipo de árvore, junto com uma análise da complexidade das operações.

O trabalho foi desenvolvido em linguagem de programação C, utilizando bibliotecas específicas para a implementação das estruturas de dados mencionadas. Os detalhes do projeto serão abordados nas seções subsequentes deste relatório. Estas incluíram discussões sobre os aspectos funcionais do programa, algoritmos empregados, chamadas de sistema relevantes e tecnologias utilizadas.

Posteriormente, serão apresentados os resultados do programa, acompanhados de medições e análises dos dados coletados, geralmente apresentados em tabelas para facilitar a compreensão. Finalmente, na conclusão, as informações apresentadas ao longo do relatório serão reforçadas.

2 Hardware utilizado

Os testes realizados foram feitos em um computador pessoal, com as seguintes especificações:

Marca	VAIO
Modelo	FE14
Sistema Operacional	UBUNTU 22.04 LTS
Processador	CORE I3 11ª GERAÇÃO
Memória Ram	20 GB DDR4

Problema 1

O programa em C implementou uma Biblioteca de Música, organizando informações por Artista, Álbum e Música. Utiliza uma árvore vermelha-preta para armazenar artistas, associando álbuns e listas ordenadas de músicas a cada artista. Permite inserção, busca e remoção de artistas, álbuns e músicas, respeitando as relações hierárquicas entre eles.

Problema 2

Repita o exercício anterior agora utilize uma árvore 2-3. Depois compare o tempo e o caminho para buscar um Artista em cada uma das árvores. Faça uma análise dos resultados. 2

Problema 3

O problema envolve a implementação de um gerenciador de memória baseado em uma árvore 4-5, onde cada nó representa um bloco de 1 Mbyte. O programa em C deve permitir o cadastro de nós da árvore, indicando se são livres ou ocupados e fornecendo os endereços iniciais e finais. Além disso, o usuário pode solicitar a alocação de uma quantidade específica de blocos, e o programa retorna as informações do nó correspondente, marcando-o como ocupado. Da mesma forma, há uma função para liberar blocos, atualizando a árvore e mantendo os nós intercalados de acordo com a situação (livre ou ocupado). O programa deve considerar a concatenação de nós adjacentes quando a situação de um nó muda e garantir que os blocos adjacentes estejam no mesmo nó após as operações.

6 Resolução

6.1 Estuturas para questão 1 As estruturas definidas no código são parte de um sistema de armazenamento de informações relacionadas a músicas, artistas e álbuns, utilizando conceitos de listas duplas e árvores rubro-negra.

• Struct Musica: Atributos: titulo[100]: Armazena o título da música. duracao: Armazena a duração da música em minutos (assumindo que seja uma unidade de tempo em minutos).

• Struct DadoArtista: Atributos: nome_artista[50]: Armazena o nome do artista. estilo_musical[50]: Armazena o estilo musical associado ao artista. num_albuns: Indica o número de álbuns lançados pelo artista.

• Struct DadoAlbum: Atributos: titulo[100]: Armazena o título do álbum. anoLancamento: Indica o ano de lançamento do álbum. qtdMusicas: Representa a quantidade de músicas presentes no álbum.

• Struct ListaDupla: Atributos: musicas: Um ponteiro para a struct Musica, representando uma lista dupla de músicas. anterior: Ponteiro para o nó anterior na lista dupla. proximo: Ponteiro para o próximo nó na lista dupla.

• Struct Album: Atributos: info: Um ponteiro para a struct DadoAlbum, armazenando informações sobre o álbum. musicas: Um ponteiro para a struct ListaDupla, representando a lista de músicas do álbum. cor: Cor associada ao nó na estrutura de árvore. esq e dir: Ponteiros para os filhos esquerdo e direito, respectivamente, na árvore.

• Struct Artista: Atributos: info: Um ponteiro para a struct DadoArtista, contendo informações sobre o artista. album: Um ponteiro para a struct Album, representando a árvore de álbuns associada ao artista. cor: Cor associada ao nó na estrutura de árvore. esq e dir: Ponteiros para os filhos esquerdo e direito, respectivamente, na árvore.

6.2 Funções para questão 1

Essas funções fornecem operações básicas para manipulação de uma Árvore rubro negra de Artista e para Album, incluindo inicialização, leitura, adição, remoção, impressão e liberação de memória. A lógica utilizada para as funções é a mesma, porém a estrutura é alterada para álbum ou para artista.

6.2.1 Função cor:

Esta função determina a cor de um nó em uma árvore, seguindo as convenções de cores em árvores rubro-negras. Se o nó for nulo, considera-se preto (BLACK); caso contrário, retorna a cor real do nó.

6.2.2 Função cria_No_Artista:

Cria e retorna um novo nó para a árvore de artistas. Inicializa o nó como nulo e retorna o ponteiro correspondente.

6.2.3 Função rotacao_esquerda_artista:

Realiza uma rotação à esquerda em um nó da árvore de artistas. Atualiza as referências dos nós envolvidos e ajusta as cores conforme as regras de árvores rubro-negras.

6.2.4 Função rotacao_direita_artista:

Efetua uma rotação à direita em um nó da árvore de artistas. Atualiza as referências dos nós envolvidos e ajusta as cores conforme as regras de árvores rubro-negras.

6.2.5 Função troca_Cor_artista:

Inverte a cor de um nó na árvore de artistas e também inverte a cor de seus filhos, se existirem.

6.2.6 Função balanceia_artista:

Realiza operações de balanceamento em um nó da árvore de artistas, garantindo a manutenção das propriedades de árvores rubro-negras.

6.2.7 Função lerInformacaoArtista:

Solicita ao usuário informações sobre um artista (nome e estilo musical) e retorna um ponteiro para uma estrutura DadoArtista com essas informações.

6.2.8 Função insere_NO_artista:

Insere um novo nó na árvore de artistas de acordo com a ordem alfabética do nome do artista. Após a inserção, realiza operações de balanceamento.

6.2.9 Função insere_RB:

Função externa que facilita a inserção de um novo artista na árvore de artistas. Retorna 1 se a inserção foi bem-sucedida e ajusta a cor da raiz para preto.

6.2.10 Função buscaArtista:

Busca um artista na árvore de artistas pelo nome e retorna um ponteiro para o nó correspondente, se encontrado.

6.2.11 Função adcionaAlbumEmArtista:

Adiciona um álbum a um artista específico, incrementando o número total de álbuns do artista.

6.2.12 Função move_esq_red:

Realiza operações para mover um nó vermelho para a esquerda, mantendo as propriedades de árvores rubro-negras.

6.2.13 Função move_dir_red:

Efetua operações para mover um nó vermelho para a direita, mantendo as propriedades de árvores rubro-negras.

6.2.14 Função remove_menor:

Remove o nó mais à esquerda da subárvore, garantindo a manutenção das propriedades de árvores rubro-negras.

6.2.15 Função procuraMenor: Encontra e retorna o nó mais à esquerda na subárvore.

6.2.16 Função buscarFolha_artista: Encontra a folha mais à direita (maior valor) na subárvore.

6.2.17 Função remove_NO_artista:

Remove um nó específico da árvore de artistas, mantendo as propriedades de árvores rubro-negras. Realiza operações de balanceamento conforme necessário.

6.2.18 Função remove_arvRB:

Função externa que facilita a remoção de um artista na árvore de artistas. Retorna 1 se a remoção foi bem-sucedida e ajusta a cor da raiz para preto.

6.2.19 Função imprimirArtista:

Imprime as informações dos artistas na árvore, incluindo nome, estilo musical, quantidade de álbuns e, se houver, informações sobre os álbuns associados. A cor do nó também é indicada (vermelha ou preta).

6.3 Funções para músicas

Essas funções fornecem operações básicas para manipulação de uma lista dupla de músicas, incluindo inicialização, leitura, adição, remoção, impressão e liberação de memória. A mesma lógica de música foi utilizada para questão 1 e 2.

6.3.1 Função inicializarLista:

Inicializa uma lista dupla de músicas e retorna um ponteiro para a lista alocada dinamicamente. A lista é inicializada com ponteiros para a música (inicialmente nulo), anterior e próximo (ambos nulos).

6.3.2 Função lerDadosMusica:

Solicita ao usuário informações sobre uma música (nome e duração) e retorna um ponteiro para uma estrutura Musical contendo essas informações.

6.3.3 Função adicionarMusica:

Adicionou uma nova música à lista dupla. Cria um novo nó para a música e o insere no final da lista, mantendo a ordem de inserção. Retorna um ponteiro para o nó recém-adicionado.

6.3.4 Função removerMusica:

Remove uma música da lista dupla com base no título fornecido. Retorna um ponteiro para o nó removido, caso exista. Se a música não for encontrada, retorna nulo. O título é usado para localizar a música na lista.

6.3.5 Função imprimirLista:

Imprime as informações de todas as músicas presentes na lista dupla. Percorre a lista e exibe o título e a duração de cada música.

6.3.6 Função liberarLista:

Libera a memória alocada dinamicamente para todos os nós da lista dupla. A função percorre a lista e libera cada nó individualmente até o final da lista. Após a execução, o ponteiro da lista é ajustado para nulo, indicando que a lista está vazia e a memória foi liberada.

6.4 Estuturas para questão 2

As estruturas definidas no código são parte de um sistema de armazenamento de informações relacionadas a músicas, artistas e álbuns, utilizando conceitos de listas duplas e árvores 2-3.

• Artista: Armazena informações sobre um artista. nome_artista: Nome do artista. estilo_musical: Estilo musical do artista. num_albuns: Número de álbuns do artista. album: Ponteiro para a estrutura do álbum associado ao artista.

• Arv23_artista: Implementa uma árvore 2-3 que armazena informações sobre artistas. info1 e info2: Ponteiros para os artistas associados a um nó. numinfo: Número de artistas armazenados no nó (pode ser 1 ou 2). esq, centro e dir: Ponteiros para os nós filhos da árvore.

• Album: Armazena informações sobre um álbum. titulo: Título do álbum. anoLancamento: Ano de lançamento do álbum. qtdMusicas: Quantidade de músicas no álbum. musicas: Ponteiro para a estrutura da música associada ao álbum.

• Arv23_album: Implementa uma árvore 2-3 que armazena informações sobre álbuns. info1 e info2: Álbuns associados a um nó. numinfo: Número de álbuns armazenados no nó (pode ser 1 ou 2). esq, centro e dir: Ponteiros para os nós filhos da árvore.

• Musica: Armazena informações sobre uma música. titulo: Título da música. duracao: Duração da música. proximo: Ponteiro para a próxima música na lista dupla. anterior: Ponteiro para a música anterior na lista dupla.

• ListaDupla: Implementa uma lista duplamente encadeada de músicas. inicio: Ponteiro para a primeira música na lista. fim: Ponteiro para a última música na lista.

6.5 Funções questão 2

Essas funções fornecem operações básicas para manipulação de uma Árvore 2-3 de Artista e para Album, incluindo inicialização, leitura, adição, remoção, impressão e liberação de memória. A lógica utilizada para as funções é a mesma, pórem a estrutura é alterada para album ou para artista. E a mesma lógica para música é utilizada, usando lista dupla.

6.5.1 Função lerDadosArtista: Solicita ao usuário que insira informações sobre um artista, como nome, estilo musical e número de álbuns. Aloca memória para uma estrutura Artista e retorna os dados inseridos.

6.5.2 Função cria_NO_Artista: Cria um nó da árvore 2-3 para armazenar informações sobre artistas. Inicializa o nó com uma única informação (info1), definindo info2 como NULL.

6.5.3 Função quebra_no: Divide um nó da árvore em dois quando ele atinge uma condição específica durante a inserção.

6.5.4 Função eh_folha_artista: Verifica se um nó da árvore é uma folha (não tem filhos). 6.5.5 Função adiciona_No_artista: Adiciona um novo nó com informações de um artista na árvore 2-3.

6.5.6 Função insere_no_artista: Insere um novo nó na árvore 2-3 com informações sobre um artista. Lida com casos de divisão de nós e redistribuição quando necessário.

6.5.7 Função RemoveMaiorInfoEsq: Remove o maior elemento da subárvore esquerda durante a exclusão de um nó.

6.5.8 Função RemoveArtista23: Remove um artista da árvore 2-3. Lida com redistribuição e casos especiais de remoção.

6.5.9 Função RedistribuiArv23Artista: Realiza redistribuição de informações quando um nó é removido.
6.5.10 Função buscarArtista: Busca um artista na árvore 2-3 com base no nome.

6.5.11 Função imprimeDadoArtista: Imprime os dados de um artista.

6.5.12 Função imprimirArtista: Imprime os dados de todos os artistas na árvore 2-3.

6.6 Estruturas para questão 3

• Info: int inicio, fim: Representa um intervalo de valores. char status: Indica o status associado ao intervalo, por exemplo, se está livre ou ocupado.

• Arv45: struct info *Info1, *Info2, *Info3, *Info4: São ponteiros para estruturas Info, representando informações associadas a intervalos. int NInfos: Indica o número de informações presentes no nó (pode variar de 1 a 4). struct arv45 *esq, *cen1, *cen2, *cen3, *dir: São ponteiros para os filhos da árvore. esq representa o filho da esquerda, dir o da direita, e cen1, cen2, cen3 são os filhos centrais.

6.6.1 Funções questão 3 6.6.2 criaInfo: Aloca memória para uma estrutura Info. Inicializa os campos inicio, fim e status da estrutura Info. Retorna um ponteiro para a estrutura Info criada.

6.6.3 criaNo: Aloca memória para um nó da árvore 4-5. Inicializa os campos do nó com base nos parâmetros fornecidos. Retorna um ponteiro para o nó criado.

6.6.4 ehFolha: Verifica se o nó fornecido é uma folha da árvore (sem filhos).

6.6.5 adicionaNo: Adiciona uma nova informação (Info) e um filho apropriado a um nó existente. Divide o nó se necessário, mantendo a propriedade da árvore 4-5

6.6.6 quebraNo: Quebra um nó quando ele está cheio e uma nova informação (Info) e um filho precisam ser inseridos. Retorna o nó que contém a maior informação (Info) e o maior filho.

6.6.7 inserir45: Inicia a inserção de uma informação (Info) na árvore 4-5. Manipula casos de inserção em um nó folha ou em um nó interno. Chama adicionaNo e quebraNo conforme necessário.

7 Resultado

Tempo de Busca Artista na 23	1.3 ms
Tempo de Busca Artista na rubro negra	1.8 ms

A análise do tempo de busca é crucial para avaliar o desempenho de estruturas de dados, como a árvore 2-3 e a árvore rubro-negra. Com base na tabela acima:

Tempo de Busca na Árvore 2-3: 1.3 ms
A árvore 2-3 é conhecida por sua eficiência em operações de busca, inserção e exclusão. O tempo de busca de 1.3 ms sugere um desempenho bastante rápido. Este resultado é positivo e sugere que a estrutura 2-3 é eficiente para recuperar informações de artistas.

Tempo de Busca na Árvore Rubro-Negra: 1.8 ms
A árvore rubro-negra também é uma estrutura de dados balanceada, mas sua eficiência pode variar dependendo da distribuição dos dados e da implementação específica. O tempo de busca de 1.8 ms é um pouco mais longo do que o da árvore 2-3, mas ainda é razoavelmente eficiente.

Comparação:
Em termos de tempo de busca, a árvore 2-3 apresentou um desempenho ligeiramente melhor em comparação com a árvore rubro-negra na amostra fornecida. No entanto, é importante ressaltar que o desempenho real pode depender de muitos fatores, incluindo o tamanho da árvore, a distribuição dos dados e a eficiência da implementação específica.

Considerações Adicionais:
Para uma avaliação mais abrangente, seria útil realizar testes com diferentes conjuntos de dados e tamanhos de árvores. Além disso, considerar outros fatores, como a complexidade de inserção e exclusão, pode ser crucial para escolher a estrutura de dados mais adequada para as necessidades do sistema.