Relatório do segundo trabalho de Estrutura de Dados II

Rayssa Dos Santos Alves

[1]Universidade Federal do Piauí (UFPI)

rayssa.alves@ufpi.edu.br

Resumo. *O objetivo deste trabalho é desenvolver um sistema de gerenciamento de vocabulários de um livro-texto de inglês, permitindo a conversão entre vocabulários Inglês-Português e Português-Inglês, além de controlar as palavras e suas equivalentes em diferentes unidades do livro. O sistema, desenvolvido em C, utiliza árvores 2-3 e árvores binárias de busca para garantir organização eficiente nas operações de busca, inserção e remoção. Cada unidade do livro contém um conjunto de palavras com traduções hierárquicas, permitindo operações como busca por unidade, tradução de palavras e remoção de palavras em ambas as línguas, mantendo a integridade das estruturas.*

1. Introdução

Estruturas de dados lineares, embora essenciais, enfrentam limitações ao lidar com grandes volumes de dados devido à sua natureza sequencial. Em listas ou arrays, a busca linear exige verificar cada elemento, tornando-se mais custosa com o aumento dos dados. Além disso, inserções e remoções podem exigir o deslocamento de vários elementos, impactando o desempenho em cenários de grande escala.

Árvores são estruturas de dados utilizadas para armazenar e recuperar dados de forma rápida e eficiente. Árvores não são lineares, ou seja, os elementos não são dispostos em formasequencial [de Dados and Wiedermann]. As árvores são estruturas de dados essenciais, permitindo representar hierarquias de forma eficiente. Diferente das estruturas lineares, elas possibilitam a organização e busca hierárquica de dados, o que é crucial em bancos de dados, onde são usadas para índices, acelerando a busca e recuperação de informações.

O presente Projeto foi desenvolvido utilizando linguagem de programação C e estruturado em três pastas separadamente visando garantir uma maior organização do código fonte de forma a facilitar seu acesso, manutenção e atualização. A Figura 1 Apresenta visualmente como estão organizadas as pastas, onde a pasta Q1 guarda todos os arquivos da questão 01 do projeto, Q2 e Q3 guardam respectivamente as questões 02 e 03 do projeto.

2. Maquina utilizada

A máquina utilizada para os testes foi configurada com especificações robustas para garantir o desempenho ideal do sistema, permitindo simular condições reais de uso e avaliar a eficiência das estruturas de dados. A tabela 1 apresenta as especificações detalhadas.

Figure 1. Caption

Modelo	Acer Nitro 5 AN515-58-58W3
Procesador	Intel Core i5-12450H
Memória RAM	16 GB DDR4
Sistema operacional	Windows 11 Home

Table 1. Especificações da máquina

3. Estruturas Gerais

3.1. Questão 1

A problemática abordada na questão 1 envolve o desenvolvimento de um sistema gerenciador de vocabulário Inglês e Português, onde devemos guardar as infomações de cada palavra proporcionando para o usuário opções como consultas e a remoção das palavras. As informações referentes as palavra em português são salvas em uma arvore de busca 2-3 e as respectivas traduções são salvas em uma ávore binária de busca.

3.1.1. Árvore 2-3

Uma árvore 2-3 é uma estrutura de dados onde cada nó pode conter zero, uma ou duas informações e possui até três filhos. Essa árvore é balanceada e garante que todas as folhas estejam no mesmo nível. Elas são usadas para armazenar e gerenciar dados ordenados de forma eficiente, permitindo inserções, remoções e buscas com um tempo de execução proporcional ao logaritmo do número de chaves, mantendo o equilíbrio da árvore mesmo durante operações de modificação conforme pode ser visto na Figura 2. As principais propriedades incluem:

- A inserção de dados só acontece quando a árvore é nula ou em uma de suas folhas.
- Folhas com duas informações devem ter obrigatoriamente zero ou três informações.
- A remoção de um nó folha com apenas uma informação acarreta em impactos diferentes dependendo de qual parte da árvore o mesmo se encontra.

Para a implementação da problemática abordada foram gerados sete arquivos contendo todo o código fonte da aplicação, visando a dividir as estruturas e funcionalidades da aplicação de forma eficiente para ser usada e atualizada na medida que novas necessidades surgirem.

1. **arv23**
   Contem todas as funções necessárias para manipular a árvore de busca 2-3

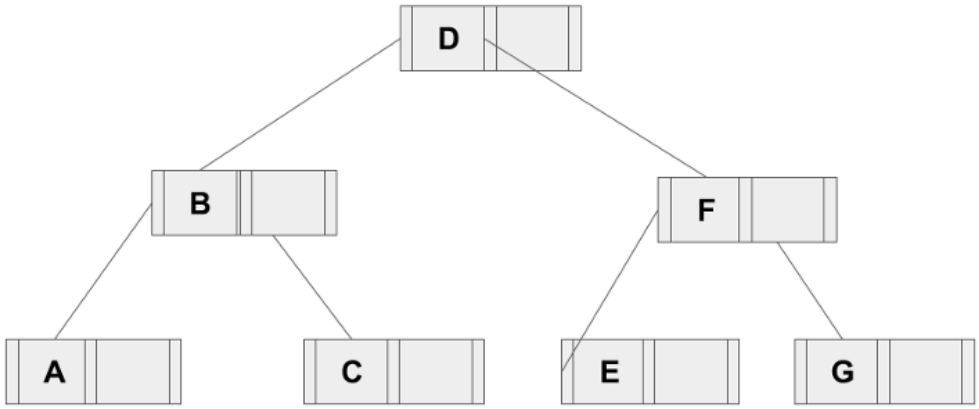

Figure 2. Arvore 2-3

2. **arvbin**
   Contem todas as funções necessárias para manipular a árvore de busca binária
3. **structs**
   Contem todas as estruturas referentes as árvores utilizadas nesse projeto
4. **lista_encadeada**
   Contem todas as funções necessárias para manipular as unidades associadas a cada nó da árvore binária.
5. **remocao**
   Contem funções de remoção de unidades, manipulando as árvores binária e 2-3 além da lista encadeada.
6. **main_teste**
   Contem as funções necessárias para realizar os testes de busca.
7. **main**
   É o arquivo principal e responsável por chamar todas as outras funções do projeto.

3.2. Questão 2

A problemática abordada na questão 2 é igual a da questão 1, porém dessa vez, foi utilizada a ávore de busca rubro negra para armazenar as palavras em português e uma árvore de busca binária para armazenar as respectivas traduções em inglês proporcionando para o usuário opções como consultas e a remoção das palavras.

3.2.1. Árvore rubro-negra

A arvore rubro negro é uma estrutura de dados em forma de árvore binária que possuem propriedades especiais para manter o balanceamento e garantir operações eficientes. Cada nó de uma árvore rubro-negra é atribuído com uma cor, vermelho ou preto, e deve seguir algumas regras para preservar o balanceamento conforme ilustrado no exemplo da Figura 3. As principais propriedades incluem:

- Cada nó é vermelho ou preto.
- A raiz é sempre preta.

- Nós vermelhos não podem ter filhos vermelhos (nenhum caminho pode ter dois nós consecutivos vermelhos).
- Todos os caminhos de um nó até suas folhas contêm o mesmo número de nós pretos.
- Nós nulos são considerados pretos.

A versão da árvore rubro negra utilizada nesse trabalho, foi pendente para a esquerda desenvolvida por Robert Sedgewick em 2008, do inglês, left leaning red black tree, garante a mesma complexidade de operações, mas possui um implementação mais simples da inserção e remoção. As principais propriedades incluem:

- Nós vermelhos aparecem apenas como filhos à esquerda.
- Nós à direita contêm apenas nós pretos.
- O equilíbrio é mantido por rotações simples e operações de alteração de cores (*flip*).

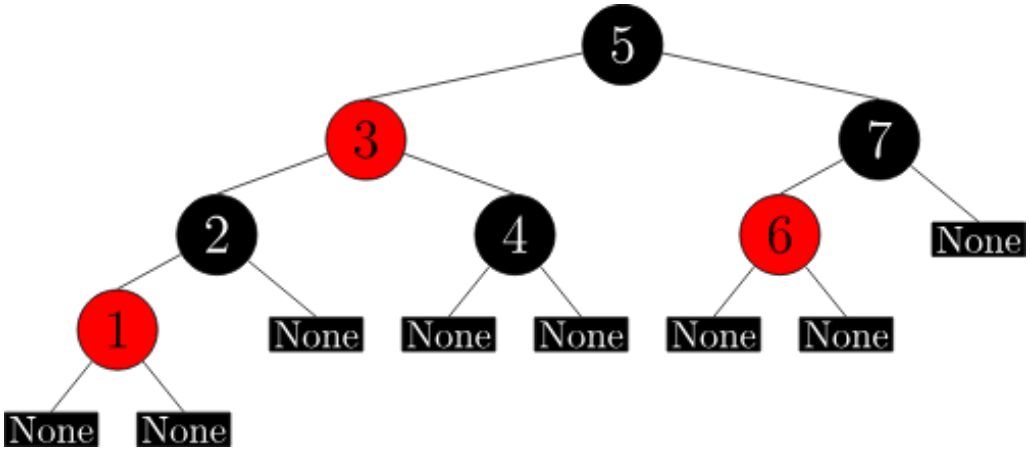

Figure 3. Left Leaning Red Black tree

Para a implementação da problemática abordada foram gerados sete arquivos contendo todo o código fonte da aplicação, visando a dividir as estruturas e funcionalidades da aplicação de forma eficiente para ser usada e atualizada na medida que novas necessidades surgirem.

1. **arvrb**
   Contem todas as funções necessárias para manipular a árvore de busca binária vermelho e preto.
2. **arvbin**
   Contem todas as funções necessárias para manipular a árvore de busca binária
3. **structs**
   Contem todas as estruturas referentes as árvores utilizadas nesse projeto
4. **lista_encadeada**
   Contem todas as funções necessárias para manipular as unidades associadas a cada nó da árvore binária.
5. **remocao**
   Contem funções de remoção de unidades, manipulando as árvores binária e 2-3 além da lista encadeada.

6. **main_teste**
   Contem as funções necessárias para realizar os testes de busca.
7. **main**
   É o arquivo principal e responsável por chamar todas as outras funções do projeto.

3.2.2. Questão 3

Na questão Q3, o cenário simulado aborda o funcionamento de um gerenciador de memória, que organiza a memória em blocos lógicos. O tamanho total da memória é definido pelo usuário, que também tem controle sobre a alocação dos blocos.

O primeiro bloco de memória é configurado manualmente, com o usuário especificando o início, o fim e o status (livre ou ocupado). Para os blocos subsequentes, o usuário define apenas o limite final de cada bloco, enquanto o sistema atribui automaticamente o status de livre ou ocupado. Esse processo continua até que toda a memória esteja completamente preenchida.

Para gerenciar os blocos de memória, foi utilizada a estrutura de árvore 2-3, conforme mencionada na questão 1. Essa abordagem permite organizar eficientemente os blocos, garantindo maior controle e desempenho na alocação e gerenciamento da memória.

Para a implementação da problemática abordada foram gerados 3 arquivos contendo todo o código fonte da aplicação, visando a dividir as estruturas e funcionalidades da aplicação de forma eficiente para ser usada e atualizada na medida que novas necessidades surgirem.

1. **arv23**
   Contem todas as funções necessárias para manipular a árvore 2-3.
2. **memoria**
   Contem todas as funções necessárias para manipular a memória.
3. **main**
   É o arquivo principal e responsável por chamar todas as outras funções do projeto.

4. Funções

4.1. Funções Árvore 2-3

adicionaChave: Essa função insere uma nova chave (informação) em um nó 2-3 que possui apenas uma chave (info1).

quebraNo: Essa função é usada para dividir um nó 2-3 que já está cheio (contém duas chaves).

arvore23_buscar_pai: Essa função encontra o nó pai de um nó.

arvore23_remover_nao_folha: Essa função remove os valores que não estão nas folhas.

possivel_remover: Essa função verifica se é possível remover um nó.

arvore23_rebalancear: Essa função balancea a árvore após a remoção, caso seja necessário.

4.2. Funções Árvore Rubro Negra

cor: Essa função verifica a cor de um nó em uma árvore vermelho-preto. Se o nó for nulo, ele é tratado como preto (cor padrão para nós nulos). Caso contrário, retorna a cor armazenada no nó.

troca_cor: Realiza a troca de cores em um nó e seus filhos. Esse processo é usado no balanceamento da árvore.

rotacao_direita: Executa uma rotação à direita em um nó da árvore vermelho-preto. É usada para corrigir desbalanceamentos e garantir que a árvore permaneça balanceada.

rotacao_esquerda: Executa uma rotação à esquerda em um nó da árvore vermelho-preto. É usada para corrigir desbalanceamentos e garantir que a árvore permaneça balanceada.

balancear: Essa função realiza o balanceamento de um nó na árvore vermelho-preto, corrigindo possíveis violações das regras da árvore.

moveEsqVermelha: Essa função é usada para mover um nó vermelho do lado direito para o lado esquerdo na subárvore atual.

moveDirVermelha: Essa função é usada para mover um nó vermelho do lado esquerdo para o lado direito na subárvore atual.

removeMenor: Remove o menor elemento da subávore à direita de uma árvore vermelho-preto.

procuraMenor: Procura o menor elemento da subávore à direita em uma árvore vermelho-preto.

4.3. Funções arvore binária

menorFilho: Essa função encontra o menor nó da subárvore a direita, ou seja, o nó mais à esquerda na subárvore, em uma árvore binária de busca, .

soUmFilho: Esta função verifica se um nó possui apenas um filho.

4.4. Principais funções

Insersão: Adiciona um novo nó a uma árvore. O nó é posicionado de acordo com as regras do tipo de árvore.

Remoção: Remove um nó de uma árvore, ajustando a estrutura para preservar suas propriedades de acordo com o tipo da árvore.

Cria nó: Esta função cria um novo nó que será inserido nas respectivas árvores.

Busca: Esta função procura por um nó específico na árvore, com base em uma chave de comparação.

Mostrar Todos As Palavras de uma unidade: Exibe todas as palavras em português armazenadas em uma unidade.

Mostrar Todas as traduções em inglês de uma palavra em portugues: Exibe todas as traduções associadas a uma palavra específica em português.

JuntarNoMemoria: Responsável por chamar as funções necessárias para juntar o nó.

Buscar menor bloco: Responsável por chamar as funções necessárias para buscar o menor bloco.

Buscar maior bloco: Responsável por chamar as funções necessárias para buscar o maior bloco.

Alocar desalocar no: Responsável por chamar as funções necessárias para alocar e desalocar o nó.

5. Resultados

Na seção de testes, são apresentados os caminhos até as palavras buscadas durante os testes em ordem crescente, seguidos de uma comparação entre a Árvore 2-3 e a Árvore Rubro-Negra, levando em consideração diferentes cenários que impactam o desempenho e a eficiência de cada estrutura. Os cenários de teste incluíram busca em ordem crescente e busca aleatória, avaliados em ambas as árvores.

A tabela 2 aprasenta todas as palavras utilizada para o esperimento.

abacaxi	aluno	amigo	avião	bolsa	cachorro
carro	casa	chave	computador	escola	felicidade
flor	floresta	gato	guerra	janela	jardim
livro	lua	mesa	oceano	porta	praia
roupas	sol	tecnologia	telefone	universidade	água

Table 2. Tabela com as palavras do experimento em ordem alfabética e reorganizada em 6 colunas

5.1. Caminho até chegar na informação

5.1.1. arvore 2-3 crescente

A Tabela 3 apresenta o caminho percorrido até encontrar a palavra buscada na Árvore 2-3.

5.1.2. Árvore Rubro-negra crescente

A Tabela 4 apresenta o caminho percorrido até encontrar a palavra buscada na Árvore Rubro-negra.

5.2. Busca crescente

O gráfico da figura 4 apresenta uma comparação do tempo de busca, em segundos, entre uma árvore 2-3 e uma árvore rubro-negra, considerando a inserção de elementos em ordem crescente. Observa-se que a árvore 2-3 apresenta um desempenho superior, com tempos de busca menores em comparação à árvore rubro-negra. Apesar de ambas as estruturas serem balanceadas, o balanceamento rigoroso da árvore 2-3 reduz sua profundidade média, o que resulta em maior eficiência para buscas realizadas em valores ordenados.

Palavra	Nível 0	Nível 1	Nível 2
abacaxi	esquerda	esquerda	esquerda
aluno	esquerda	esquerda	
amigo	esquerda	esquerda	centro
avião	esquerda		
bolsa	esquerda	centro	esquerda
cachorro	esquerda	centro	
carro	esquerda	centro	centro
casa			
chave	centro	esquerda	esquerda
computador	centro	esquerda	
escola	centro	esquerda	centro
felicidade	centro		
flor	centro	centro	esquerda
floresta	centro	centro	
gato	centro	centro	centro
guerra			
janela	direita	esquerda	esquerda
jardim	direita	esquerda	
livro	direita	esquerda	centro
lua	direita		
mesa	direita	centro	esquerda
oceano	direita	centro	
porta	direita	centro	centro
praia	direita		
roupas	direita	direita	esquerda
sol	direita	direita	
tecnologia	direita	direita	centro
telefone	direita	direita	
universidade	direita	direita	direita
água	direita	direita	direita

Table 3. Caminho da árvore 2-3

5.3. Busca decrescente

O gráfico da figura 5 compara o tempo de busca entre uma árvore 2-3 e uma árvore rubro-negra quando os elementos são inseridos em ordem aleatória. Observa-se que a árvore 2-3 apresenta tempos de busca menores, devido ao seu balanceamento mais rigoroso, que mantém alturas menores e uma estrutura mais eficiente para buscas. Essa vantagem, embora sutil, demonstra que a árvore 2-3 é mais consistente e rápida em cenários de buscas aleatórias, evidenciando sua eficiência em situações de operações imprevisíveis.

6. Conclusão

A Árvore 2-3 teve desempenho superior à Árvore Rubro-Negra em cenários de busca crescente, apresentando tempos de execução menores. Isso se deve à sua estrutura com

Palavra	Nível 0	Nível 1	Nível 2	Nível 3
abacaxi	esquerda	esquerda	esquerda	esquerda
aluno	esquerda	esquerda	esquerda	
amigo	esquerda	esquerda	esquerda	direita
avião	esquerda	esquerda		
bolsa	esquerda	esquerda	direita	esquerda
cachorro	esquerda	esquerda	direita	
carro	esquerda	esquerda	direita	direita
casa	esquerda			
chave	esquerda	direita	esquerda	esquerda
computador	esquerda	direita	esquerda	
escola	esquerda	direita	esquerda	direita
felicidade	esquerda	direita		
flor	esquerda	direita	direita	esquerda
floresta	esquerda	direita	direita	
gato	esquerda	direita	direita	direita
guerra				
janela	direita	esquerda	esquerda	esquerda
jardim	direita	esquerda	esquerda	
livro	direita	esquerda	esquerda	direita
lua	direita	esquerda		
mesa	direita	esquerda	direita	esquerda
oceano	direita	esquerda	direita	
porta	direita	esquerda	direita	direita
praia	direita			
roupas	direita	direita	esquerda	esquerda
sol	direita	direita	esquerda	
tecnologia	direita	direita	esquerda	direita
telefone	direita	direita		
universidade	direita	direita	direita	esquerda
água	direita	direita	direita	

Table 4. Caminho da árvore rubro-negra

múltiplos filhos e balanceamento eficiente, que facilita buscas sequenciais, enquanto a Árvore Rubro-Negra requer mais tempo devido às rotações adicionais necessárias para manter o balanceamento.

Em buscas aleatórias, a Árvore 2-3 também apresentou melhor desempenho, com tempos menores em comparação à Árvore Rubro-Negra. A aleatoriedade dos dados aumenta a complexidade das operações para estruturas como a Árvore Rubro-Negra, que depende de balanceamento dinâmico, enquanto a Árvore 2-3, com sua estrutura estática, se mostrou mais eficiente independentemente do padrão de dados.

A Árvore 2-3 é mais adequada para aplicações com foco em buscas rápidas, tanto ordenadas quanto aleatórias, enquanto a Árvore Rubro-Negra pode ser mais eficiente em cenários com inserções e deleções frequentes. A escolha entre as duas estruturas depende

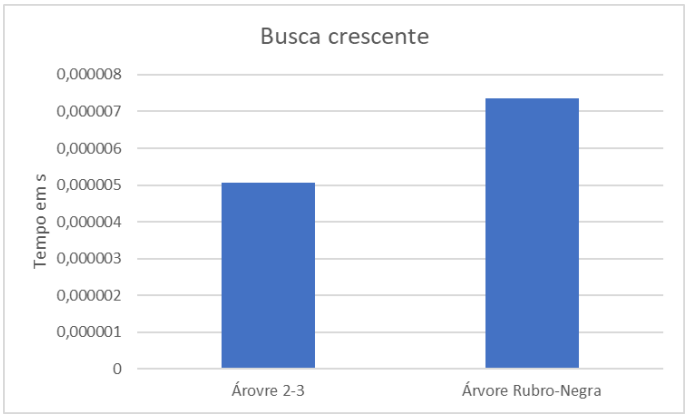

Figure 4. Busca Crescente

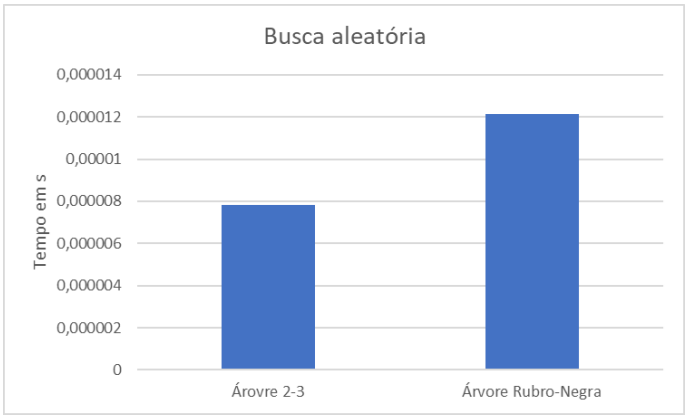

Figure 5. Busca Aleatória

das características e necessidades específicas do problema, equilibrando desempenho e flexibilidade.

References

de Dados, D. d. E. and Wiedermann, L. T. Árvores.