Conversão de Vocabulários Bilíngues com Árvores 2-3 e Rubro-Negra e Alocação de Memória na 2-3

Daniel Rodrigues de Sousa

1 Resumo

Este trabalho aborda a organização e manipulação de vocabulários bilíngues (Inglês-Português) em livros-texto, utilizando árvores 2-3 e Rubro Negra já as árvores binárias de busca para estruturar os dados de forma eficiente. As palavras em Português foram armazenadas em uma árvore 2-3 e na Rubro Negra, enquanto suas equivalentes em Inglês, organizadas por unidade, foram armazenadas em árvores binárias de busca. O sistema permite consultas, remoções e atualizações dinâmicas, garantindo que palavras redundantes sejam eliminadas de ambas as estruturas. Foram realizados experimentos para buscar 30 palavras em Português, registrando os caminhos percorridos e os tempos de execução.

2 Introdução

A área de Estruturas de Dados é essencial na ciência da computação, sustentando silenciosamente diversas aplicações no mercado. Apesar de seu funcionamento ser invisível para o usuário final, a ausência de uma implementação eficiente pode resultar em sistemas lentos e suscetíveis a falhas. A escolha adequada da estrutura de dados é fundamental para garantir o bom desempenho de uma aplicação e deve ser feita com base em uma análise criteriosa das necessidades do sistema.

Neste artigo, exploramos o uso de três tipos de árvores, com destaque para as árvores 2-3 e Rubro-Negra, que desempenham papéis centrais em nosso estudo de caso. Essas estruturas trabalham em conjunto com a Árvore Binária de Busca, responsável por armazenar as palavras equivalentes em Inglês. Enquanto isso, as árvores 2-3 e Rubro-Negra organizam todas as palavras em Português, juntamente com as unidades às quais pertencem.

Essa combinação de estruturas de dados possibilita a criação de uma aplicação eficiente, organizada por unidades, onde cada unidade contém um vocabulário de palavras em Português e seus respectivos significados em Inglês. O principal objetivo é garantir buscas rápidas e precisas, oferecendo uma solução robusta e escalável para a organização e manipulação de vocabulários bilíngues.

A aplicação receberá um arquivo de entrada denominado *input.txt*, que será a fonte de dados para o programa, eliminando a necessidade de interação direta com o usuário durante a execução. É essencial que este arquivo esteja formatado rigorosamente de acordo com o padrão definido na documentação proposta, garantindo que a leitura dos dados ocorra de maneira consistente e sem erros.

Além do estudo comparativo entre as duas árvores, foi implementado um código destinado ao gerenciamento de blocos de memória, utilizando a estrutura da Árvore 2-3. Esse tema será abordado de forma mais detalhada na seção de metodologia.

3 Seções Especificas

Neste tópico, serão discutidas as funções essenciais do sistema, incluindo o hardware utilizado e as estruturas necessárias para o entendimento do artigo. Cada função será detalhada quanto a seus objetivos e papel, desde

o cadastro de valores até a inserção nas estruturas das Árvore Rubro Negra e 2-3.

A análise mostrará como essas funções impactam o desempenho do sistema, abrangendo desde operações básicas até algoritmos complexos, como inserção e balanceamento. Isso permitirá ao leitor entender a interação entre as funções e como a escolha das estruturas de dados afeta diretamente a eficiência e manutenção do sistema.

3.1 Árvore Rubro Negra

A árvore Rubro-Negra é uma estrutura de dados binária de busca balanceada que oferece eficiência em operações como busca, inserção e remoção. Cada nó da árvore possui uma cor (vermelho ou preto), e o balanceamento é mantido por regras como: a raiz é sempre preta, nós vermelhos não podem ter filhos vermelhos, e todos os caminhos da raiz até as folhas contêm o mesmo número de nós pretos. Essas propriedades garantem que a altura da árvore seja limitada, permitindo operações rápidas e consistentes.

3.2 Árvore 2-3

A árvore 2-3 é uma estrutura de dados balanceada em que cada nó pode conter uma ou duas chaves e ter dois ou três filhos. Todas as folhas estão no mesmo nível, mantendo o balanceamento automático. As chaves em cada nó estão ordenadas, e os filhos são organizados de forma que os valores à esquerda sejam menores, os intermediários ao centro e os maiores à direita. Quando um nó excede a capacidade de duas chaves durante uma inserção, ele é dividido, promovendo uma chave ao nó pai para preservar a estrutura.

3.3 Informações Técnicas

Abaixo, segue as informações do hardware utilizados para os testes em ambas as árvores:

Componente	Descrição
Processador	10th Gen Intel(R) Core(TM) i5-10300H @ 2.50GHz Até 4.50 GHz
Memória RAM	8GB DDR4
Sistema Operacional	Windows 11 23H2

Table 1: Especificações de hardware e software

3.4 Estruturas da Arvore 2-3

Abaixo, serão apresentadas as estruturas utilizadas para a Árvore 2-3:

3.4.1 Árvore Binária (Palavras em Inglês)

```
typedef struct EnglishBinTree {
    LinkedUnitList *units;
    char *englishWord;
    struct EnglishBinTree *right, *left;
} EnglishBinTree;
```

Listing 1: Struct Palavras em Inglês

3.4.2 Árvore 2-3

```
typedef struct translationInfo {
    char *portugueseWord;
    EnglishBinTree *englishWord;
} TranslationInfo;
```

Listing 2: Informação usada pelas árvores

```
typedef struct Portuguese23Tree {
    TranslationInfo info1;
    TranslationInfo info2;
    struct Portuguese23Tree *left, *middle, *right;
    int infoCount;
} Portuguese23Tree;
```

Listing 3: Árvore 2-3

3.4.3 Lista Encadeada

```
typedef struct linkedUnitList {
  char *unitName;
  struct linkedUnitList *next;
} LinkedUnitList;
```

Listing 4: Lista Encadeada com as Unidades

3.4.4 Estruturas da Árvore 2-3: Gerenciamento de Memória

```
typedef struct Info {
    int start, end, status;
} Info;
```

Listing 5: Começo Final e Status

```
typedef struct Memory {
    Info *info1;
    Info *info2;
    struct Memory *left, *center, *right;
    int numKeys;
} Memory;
```

Listing 6: Árvore 2-3 do Gerenciamento de Memória

3.4.5 Estruturas da Árvores Rubro Negra

Abaixo, serão apresentadas as estruturas utilizadas para o funcionamento da aplicação:

3.4.6 Arvore Binária (Palavras em Inglês)

```
typedef struct infoEngPT {
  char *englishWord;
  Units *units;
```

```
} InfoEngPT;
```

Listing 7: Palavras em Inglês

```
typedef struct engPT {
  InfoEngPT info;
  struct engPT *left, *right;
} EngPT;
```

Listing 8: Árvore Binária de Busca

3.4.7 Arvore Rubro Negra

```
typedef enum color {
  RED = 0,
  BLACK = 1
} Color;
```

Listing 9: Cores da Rubro-Negra

```
typedef struct infoBin {
  EngPT *binaryTreeEnglish;
  char *portugueseWord;
} InfoBin;
```

Listing 10: Informações para a Rubro-Negra

```
typedef struct redBlackTreeNode {
  RBTreeNodeInfo info;
  Color color;
  struct redBlackTreeNode *left, *right;
} RedBlackTreeNode;
```

Listing 11: Árvore Rubro-Negra

3.5 Funções

Nessa seção será apresentada os escopos das funções utilizadas, para ambas árvores, com descrições breves referentes a cada respectiva função.

3.5.1 Função de Inserir

As funções **insertIntoRBTree** e **insertPortugueseWord** lêem palavras de um arquivo .txt e as insere em árvores rubro-negras, mantendo sua estrutura balanceada. Essa organização garante buscas rápidas e eficientes, sendo ideal para sistemas de tradução ou dicionários.

```
int insertIntoRBTree(RBTree **root, InfoBin info);
```

```
int insertPortugueseWord(Portuguese23Tree **tree, char *portugueseWord, char *englishWord
    , char *unit);
```

3.5.2 Função de Exibir por Unidade

As funções **DisplayUnit** e **displayWordsForUnit** são responsáveis por exibir todas as palavras de uma unidade específica informada pelo usuário. Ela exibirá tanto a palavra em português, como sua equivalente em inglês.

```
void displayUnit(RBTree *root, int unit, int *found);
```

```
void displayWordsForUnit(Portuguese23Tree *tree, char *unit);
```

3.5.3 Busca de Palavras em Inglês por Palavras em Português

As funções **findEnglishByPortuguese** e **displayTranslationPortuguese** são projetadas para realizar a busca de uma palavra específica em português e identificar todas as palavras em inglês que estão relacionadas a ela.

```
void findEnglishByPortuguese(RBTree *node, const char *portugueseWord);
```

```
void displayTranslationPortuguese(Portuguese23Tree **root, const char *portugueseWord);
```

3.5.4 Função Remover por Palavras em Inglês

As funções **removeFromRBTreeEN** e **removeEnglishFindUnit** removem uma palavra em inglês e suas correspondentes traduções em português.

```
void removeFromRBTreeEN(RBTree **root, const char *englishWord, int unit);
```

```
int removeEnglishFindUnit(Portuguese23Tree **root, const char *englishWord, const char *
    unit);
```

3.5.5 Função Remover por Palavras em Português

As funções **removeFromRBTreeEN** e **removePortugueseWordFindUnit** removem uma palavra em português e suas correspondentes traduções em inglês.

```
void removeFromRBTreePT(RBTree **root, char *portugueseWord, int unit);
```

```
int removePortugueseWordFindUnit(Portuguese23Tree **root, char *portugueseWord, const
    char *unit);
```

3.5.6 Funções do Gerenciamento de Memória

3.6 Função de Inserir

Essa função é chamada no menu principal da aplicação, ela será responsável por inserir na árvores as informações do nó, como: Inicio, Fim, Status (0 - Ocupado, 1 - Livre)

```
Memory *insertTree23(Memory **raiz, Info info, Memory *pai, Info *promove);
```

3.6.1 Alocar e Desalocar Nós

Os nós são inicialmente definidos como livre ou ocupados, mas ao decorrer dessa interação com o usuários isso pode mudar, com a função:

```
void allocateAndDesallocate(Memory **tree, int quantNodes, int status);
```

3.7 Exibir Nós

Por fim, a programa deverá mostrar todos os nós em ordem, com suas respectivas informações.

```
1 void displayInfos(Memory *root);
```

3.7.1 Observação

As funções mencionadas acima representam apenas uma pequena parte de um complexo conjunto de funções auxiliares que operam de maneira integrada. Esse trabalho em conjunto é essencial para garantir que as funções principais sejam executadas com eficiência e eficácia, maximizando o desempenho do sistema como um todo.

4 Metodologia

A metodologia consistiu em comparar o desempenho de busca entre árvores 2-3 e Rubro-Negra em um ambiente controlado, utilizando a mesma máquina para garantir consistência nos resultados. Foram realizadas buscas em uma sequência fixa de 30 palavras, selecionadas aleatoriamente e distribuídas uniformemente entre os níveis das árvores, assegurando condições iguais de execução.

Ambas as árvores foram construídas com cerca de 400 palavras, organizadas conforme os princípios de cada estrutura. A árvore 2-3 utiliza balanceamento dinâmico com 2 a 3 chaves por nó, enquanto a Rubro-Negra emprega regras baseadas em cores para manter o balanceamento. Durante o experimento, registrou-se o tempo de busca para cada palavra, permitindo uma análise comparativa de eficiência.

Adicionalmente, um arquivo reduzido (*input.txt*) com 7 palavras foi utilizado para uma análise detalhada e validação dos algoritmos em condições específicas. Essa abordagem permitiu mapear resultados e identificar possíveis inconsistências. A metodologia garante uma avaliação objetiva das estruturas, oferecendo subsídios para escolha em cenários que demandam alta performance em consultas.

Por fim, foi desenvolvido um programa para o gerenciamento de memória utilizando a Árvore 2-3. O programa está estruturado em três funcionalidades principais: inicialmente, solicita o tamanho total do bloco de memória e realiza perguntas para cada bloco até atingir o limite especificado pelo usuário. Durante essa etapa, os blocos são inseridos sequencialmente, alternando entre os estados de ”Livre” e ”Ocupado”. Em seguida, o usuário pode realizar ações como alocar blocos previamente marcados como ”Livres” ou liberar blocos ocupados. Por fim, o programa permite monitorar o desempenho e o comportamento geral do sistema, exibindo informações detalhadas sobre o estado atual da memória.

5 Resultados

Na figura abaixo, há uma prévia dos resultados obtidos, sendo um comparativo de tempo em segundos entre as duas árvores, nas seções abaixo será melhor detalhado.

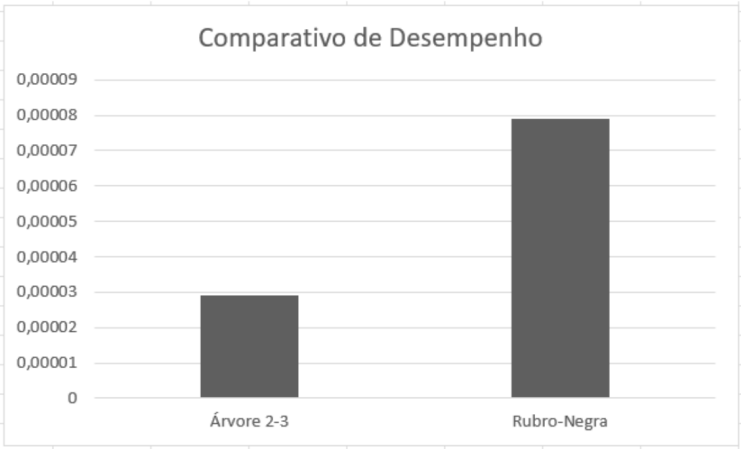

Figure 1: Comparativo entre as árvores 2-3 e Rubro-Negra

A seguir, serão apresentados os resultados obtidos para os tempos de execução coletados das árvores Rubro-Negra e 2-3. Esses dados estão dispostos em tabelas e figuras que ilustram, de forma detalhada, as comparações entre as duas estruturas em diferentes condições de teste. As tabelas sintetizam os tempos médios de operação. Também, será listado o caminho percorrido para chegar em cada informação.

5.1 Árvore Rubro Negra

Nessa seção, serão apresentados os resultados referentes aos tempos de busca, bem como os resultados referentes aos caminhos percorridos para as buscas.

5.1.1 Metrificação

Abaixo, apresentamos a Tabela 2. Para a coleta dos resultados, foram realizadas buscas utilizando as palavras listadas na tabela. O objetivo foi medir os tempos médios de resposta e avaliar a eficiência do sistema em relação aos termos pesquisados. Essa análise fornece uma visão clara do desempenho em diferentes cenários, conforme detalhado a seguir.

Média de tempo	**Palavras buscadas**
0,000079s	indicação, melhorar, recuperação, segmento, ventilador, condensar, carregador, textura, colapso, motor, polimorfismo, barramento, característica, saudação, interruptor, otimizar, compilar, rodar, cache, transporte, assíncrono, depurar, analisar, interpretar, monte, pixel, informações, entrada, roteador, guia

Table 2: Tabela de tempos médios de resposta para as palavras buscadas.

5.1.2 Caminho Percorrido

Primeiramente, é importante destacar que existem diferentes tipos de percursos em árvores, como: *pré-ordem*, *in-ordem* e *pós-ordem*. O percurso utilizado neste caso foi o de *pré-ordem*, no qual a visita ocorre na seguinte sequência: primeiro o nó raiz (informação principal), seguido pelos nós à esquerda até que não haja mais informações, e por fim, os nós à direita. Esse conceito será detalhado com o auxílio do *output* gerado pelo código, bem como das figuras elaboradas pelo autor.

O bloco de notas *input.txt* trás as palavras em português (que foram as chaves de ordenação) nessa ordem: latência, vazão, protocolo, saudação, alcance, moldura e seção. Essa inserção gerou uma árvore vermelha e preta completamente balanceada, como podemos ver na imagem abaixo

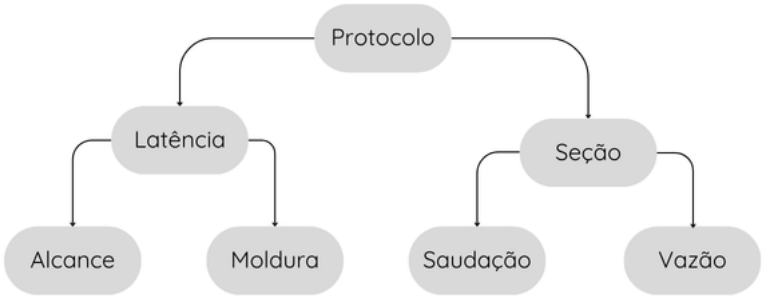

Figure 2: Árvore Rubro Negra gerada após a inserção das palavras

Foi desenvolvida uma função no código com o propósito de exibir as palavras na ordem exata em que foram percorridas durante o processo de busca. Essa função permite acompanhar o fluxo de execução e visualizar a sequência de palavras conforme são processadas. O resultado gerado por essa função é apresentado a seguir.

```
Palavra: protocolo, Cor: Preto
    Esquerda ->
    Palavra: latência, Cor: Preto
        Esquerda ->
        Palavra: alcance, Cor: Preto
            Esquerda ->
            NULL
            Direita ->
            NULL
        Direita ->
        Palavra: moldura, Cor: Preto
            Esquerda ->
            NULL
            Direita ->
            NULL
    Direita ->
    Palavra: seção, Cor: Preto

=== Palavra encontrada: seção ===
Unidades associadas:
        Esquerda ->
        Palavra: saudação, Cor: Preto
            Esquerda ->
            NULL
            Direita ->
            NULL
        Direita ->
        Palavra: vazão, Cor: Preto
            Esquerda ->
            NULL
            Direita ->
            NULL
```

Observe que a sequência em que as palavras foram listadas sugere que a leitura foi realizada em *pré-ordem*. Por

exemplo, ao buscar pela palavra *seção*, o programa começará encontrando o nó *Protocolo*, seguindo para a direita até alcançar os nós *Latência* e *Alcance*. Quando chegar a *NULL*, o programa retorna à última pendência, exibindo o valor *Moldura*, e, finalmente, chega à palavra *seção*. Esse processo de leitura pode ser visualizado na figura abaixo.

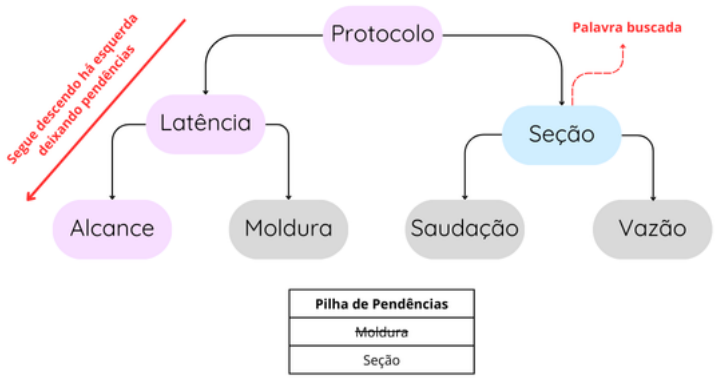

Figure 3: Árvore Rubro Negra: busca em pré-ordem

Percebe-se, que há pendências, que permite com que o programa enxergue as palavras para que quando volte às exiba, isso foi especificado na imagem, que moldura e seção ficaram como pendências e posteriormente foram lidas. Uma pequena observação, é que na pré-ordem há pendência tanto para opreções de movimento para esquerda quando para direita, mas por fins didáticos na figura a tabela de pendência só foi armazenada com as pendências da direita.

5.2 Árvore 2-3

5.2.1 Metrificação

Abaixo, apresentamos a Tabela 3. Para a coleta dos resultados, foram realizadas buscas utilizando as palavras listadas na tabela. O objetivo foi medir os tempos médios de resposta e avaliar a eficiência do sistema em relação aos termos pesquisados. Essa análise fornece uma visão clara do desempenho em diferentes cenários, conforme detalhado a seguir.

Média de tempo	**Palavras buscadas**
0,000023s	indicação, melhorar, recuperação, segmento, ventilador, condensar, carregador, textura, colapso, motor, polimorfismo, barramento, característica, saudação, interruptor, otimizar, compilar, rodar, cache, transporte, assíncrono, depurar, analisar, interpretar, monte, pixel, informações, entrada, roteador, guia

Table 3: Tabela de tempos médios de resposta para as palavras buscadas.

5.2.2 Caminho Percorrido

Novamente, foi feita a busca em *pré-ordem*. Abaixo serão disponibilizados os resultados esperados, bem como a saída gerada pelo código.

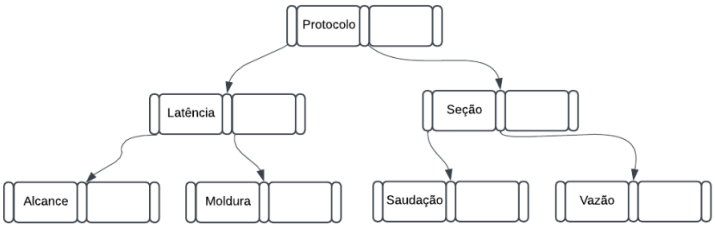

Figure 4: Árvore 2-3 gerada após a inserção

Como no exemplo anterior, segue abaixo a árvore gerada:

```
Palavra: protocolo
Esquerda ->
    Palavra: latência
    Esquerda ->
        Palavra: alcance
        Esquerda ->
            NULL
    Centro ->
        Palavra: moldura
        Esquerda ->
            NULL
Centro ->
    Palavra: seção
    Esquerda ->
        Palavra: saudação
        Esquerda ->
            NULL
    Centro ->
        Palavra: vazão
        Esquerda ->
            NULL
```

6 Conclusão

Neste relatório, foram analisados os casos de uso das árvores 2-3 e Rubro-Negra, com ênfase especial no desempenho das operações de busca. A árvore 2-3 demonstrou um desempenho superior em termos de agilidade e estabilidade nas buscas, o que é justificável devido ao seu rigoroso processo de balanceamento. Esse balanceamento mais restritivo contribui para um desempenho mais eficiente nas operações de busca, tornando a árvore 2-3 mais eficaz para cenários em que esse tipo de operação é crítico.

Contudo, a implementação da árvore 2-3 é significativamente mais complexa. O alto grau de rigor no balanceamento torna a implementação de operações como a remoção ou a travessia em pré-ordem consideravelmente mais desafiadora. Esse aspecto implica em um aumento substancial na complexidade do código e no tempo de desenvolvimento.

Em resumo, embora a árvore 2-3 apresente um desempenho superior nas buscas em comparação com a árvore Rubro-Negra, é essencial considerar os desafios de sua implementação. A escolha entre as duas estruturas de dados deve ser feita com base nas necessidades específicas do sistema em questão, levando em consideração o equilíbrio entre desempenho e complexidade de implementação.