Comparação entre Árvores ABP e AVL para a Geração de Paráfrases

Disciplina de Estrutura de Dados - INF01203

Relatório do Trabalho Final

Autores: Thomas Henrique Schmitz - 00588622
e Eduardo Henrique Drumm Menezes - 00588224
Docente: Viviane Pereira Moreira

Universidade Federal do Rio Grande do Sul
Instituto de Informática

Porto Alegre, 10 de janeiro de 2025

Conteúdo

1 Introdução

A geração de paráfrases é uma técnica importante em processamento de linguagem natural e possui diversas aplicações, como reformulação de texto, tradução automática e compressão textual. Neste trabalho, utilizamos uma Árvore Binária de Pesquisa (ABP) e a AVL como estruturas de dados para realizar a tarefa de substituição de palavras por sinônimos, permitindo a reformulação eficiente de textos.

Este relatório apresenta uma explicação teórica sobre as ABPs e AVLs, detalhes sobre sua implementação para a tarefa de geração de paráfrases e uma análise de desempenho baseada em um conjunto de experimentos.

2 Árvore Binária de Pesquisa (ABP)

2.1 Definição

Uma Árvore Binária de Pesquisa é uma estrutura de dados hierárquica que organiza seus elementos de forma que:

- Para cada nó, os elementos na subárvore à esquerda são menores que a chave do nó.
- Os elementos na subárvore à direita são maiores que a chave do nó.

Essa propriedade permite realizar operações como busca, inserção e remoção em tempo médio de $O(\log n)$ em árvores balanceadas. Isso significa que, no pior caso, todas as operações sobre a ABP consomem tempo proporcional à altura da árvore. No caso deste experimento, a árvore binária de pesquisa foi organizada na ordem lexicográfica das palavras, ou seja, a ordem que elas aparecem no dicionário.

2.2 Operações Fundamentais

- **Busca**: Percorre a árvore comparando a chave alvo com os nós, movendo-se para a subárvore esquerda ou direita, conforme necessário.
- **Inserção**: Para adicionar novos nodos na ABP, é preciso começar do nó-raiz e descer de forma recursiva, procurando o local certo para inserir o novo nó.
- **Remoção**: Ao remover um nodo da árvore, é necessário manter as propriedades da árvore, para isso, é necessário pegar o maior elemento da subárvore esquerda, caso ela exista, do nodo a ser removido e substituir no nodo original removido.

2.3 Aplicações

As ABPs são amplamente utilizadas em sistemas de busca, compressão de dados e algoritmos que exigem acesso eficiente a pequenos conjuntos de dados. Ela possui a vantagem de não precisar realizar rotações, mas isso pode trazer um menor desempenho em suas operações fundamentais ao comparar com outros modelos de árvore, uma vez que, quando houver dados em larga escala, teremos mais comparações para realizar cada ação.

2.4 Implementação da Geração de Paráfrases

A tarefa de geração de paráfrases consiste em substituir palavras de um texto original por sinônimos encontrados em um dicionário. A implementação utiliza uma ABP para armazenar o dicionário, onde:

- A chave é a palavra original.
- O valor associado é o sinônimo correspondente.

2.4.1 Estrutura de Dados

Cada nó da árvore armazena:

- **Chave:** Palavra do texto original.
- **Valor:** Palavra sinônima.
- **Filho Esquerdo e Direito:** Ponteiros para subárvores.

2.4.2 Funcionamento do Algoritmo

O funcionamento do algoritmo para geração de paráfrases é dividido em três etapas principais: carregamento do dicionário, processamento do texto e escrita do texto parafraseado. A seguir, cada etapa é descrita detalhadamente com exemplos de código.

1. **Carregamento do Dicionário:**
   - O dicionário é carregado a partir de um arquivo de texto onde cada linha contém uma palavra e seu sinônimo, separados por espaço. Um exemplo do formato do arquivo pode ser visto abaixo:

     ```
     palavra1 sinonimo1
     palavra2 sinonimo2
     ...
     ```

   - Cada entrada do dicionário é inserida na Árvore Binária de Pesquisa (ABP) utilizando a função `inserir_ABP`. A função insere as palavras considerando a ordem lexicográfica:
     - Palavras que vêm antes no alfabeto são posicionadas à esquerda.
     - Palavras que vêm depois no alfabeto são posicionadas à direita.

O trecho de Código 1 abaixo ilustra a função de inserção na ABP das palavras do dicionário:

```c
pNodoA* inserir_ABP(pNodoA *raiz, char *chave, char *sinonimo) {
    if (raiz == NULL) {
        pNodoA *novo = malloc(sizeof(pNodoA));
        novo->chave = strdup(chave);
        novo->sinonimo = strdup(sinonimo);
        novo->esq = novo->dir = NULL;
        return novo;
    }
    if (strcmp(chave, raiz->chave) < 0) {
        raiz->esq = inserir_ABP(raiz->esq, chave, sinonimo);
    } else if (strcmp(chave, raiz->chave) > 0) {
        raiz->dir = inserir_ABP(raiz->dir, chave, sinonimo);
    }
    return raiz;
}
```

Código 1: Código usado para inserir na árvore ABP

2. **Processamento do Texto:** O texto de entrada é processado palavra por palavra. Cada palavra é normalizada (removendo pontuações e convertendo para minúsculas) e buscada na ABP. Se um sinônimo for encontrado, ele substitui a palavra original; caso contrário, a palavra original é mantida. O Código 2 abaixo exemplifica a lógica de processamento:

```c
void parafrasear(FILE *entrada, FILE *saida, pNodoA *dicionario) {
    char palavra[100];
    int c;

    while ((c = fgetc(entrada)) != EOF) {
        if (isspace(c)) {
            // Preserva espa os no texto
            fputc(c, saida);
        } else {
            ungetc(c, entrada);
            // L  uma palavra at  encontrar um espa o ou pontua  o
            fscanf(entrada, "%99s", palavra);
            char pontuacao = '\0';
            int len = strlen(palavra);

            // Verifica se o  ltimo  caractere  uma pontua  o
            if (len > 0 && ispunct(palavra[len - 1])) {
                pontuacao = palavra[len - 1];
                palavra[--len] = '\0';  // Remove a pontua  o temporariamente
            }

```

```
22              // Converte a palavra para min scula
23              for (int i = 0; palavra[i]; i++) {
24                  palavra[i] = tolower((unsigned char)palavra[
                        i]);
25              }
26
27              // Procura a palavra no dicion rio
28              pNodoA *sinonimo = consulta_ABP(dicionario,
                    palavra);
29
30              if (sinonimo) {
31                  fprintf(saida, "%s", sinonimo->sinonimo);
32              } else {
33                  fprintf(saida, "%s", palavra);
34              }
35              if(pontuacao == '-') putc('-', saida);
36          }
37      }
38  }
```

Código 2: Código usado para parafrasear utilizando árvore ABP

A normalização de palavras assegura que variações de maiúsculas e minúsculas não afetem a busca. Além disso, algumas pontuações específicas, como travessões, são preservadas no texto de saída. A função `consultaABP` é conhecida e foi trazida dos requisitos do trabalho. O Código 3 abaixo consulta a ABP:

```
1   pNodoA* consulta_ABP(pNodoA *raiz, char *chave) {
2       while (raiz != NULL) {
3           comparacoes++;
4           if (!strcmp(raiz->chave, chave)) {
5               comparacoes++;
6               return raiz;
7           } else {
8               comparacoes++;
9               if(strcmp(raiz->chave, chave) > 0) raiz = raiz->
                    esq;
10              else raiz = raiz->dir;
11          }
12      }
13      return NULL;
14  }
```

Código 3: Código usado para consultar árvore ABP

3. **Escrita do Texto Parafraseado:** Após processar cada palavra, o texto reformulado é salvo em um arquivo de saída. A função `fprintf` é utilizada para escrever tanto palavras substituídas quanto palavras originais, preservando espaços e formatação do texto.

   O Código 4 também assegura que as estatísticas da ABP (como número de nós e altura) sejam registradas para análise de desempenho:

```
1
2     void salvar_estatisticas_ABP(const char *
          arquivo_estatisticas, const char *arquivo_entrada,
          const char *arquivo_dicionario, pNodoA *raiz) {
3         FILE *fp = fopen(arquivo_estatisticas, "w");
4         if (!fp) {
5             perror("Erro ao abrir o arquivo de estatisticas"
                  );
6             return;
7         }
8
9         fprintf(fp, "======== ESTATISTICAS ABP
              ============\n");
10        fprintf(fp, "Arq Entrada: %s\n", arquivo_entrada);
11        fprintf(fp, "Arq Dicionario: %s\n",
              arquivo_dicionario);
12        fprintf(fp, "Numero de Nodos: %d\n",
              contar_nodos_ABP(raiz));
13        fprintf(fp, "Altura: %d\n", altura_ABP(raiz));
14        fprintf(fp, "Rotacoes: 0\n");
15        fprintf(fp, "Comparacoes: %d\n", comparacoes);
16        fclose(fp);
17    }
```

Código 4: Código usado para salvar estatísticas da árvore ABP

Essa abordagem modular e eficiente permite que o algoritmo seja aplicado em diferentes contextos de geração de paráfrases, garantindo flexibilidade e escalabilidade.

3 Árvore AVL

3.1 Definição

A Árvore AVL é uma árvore binária de busca balanceada, onde a diferença de altura entre as subárvores esquerda e direita de qualquer nó é no máximo 1. Esse balanceamento é mantido por meio de rotações (simples ou duplas) realizadas durante as operações de inserção e remoção.

A propriedade de balanceamento garante que as operações fundamentais, como busca, inserção e remoção, tenham complexidade $O(\log n)$ mesmo no pior caso. Isso torna a Árvore AVL uma estrutura eficiente para armazenar e acessar grandes conjuntos de dados.

3.2 Operações Fundamentais

As operações fundamentais em uma Árvore AVL incluem:

- **Busca**: Semelhante à ABP, percorre a árvore comparando a chave alvo com os nós.
- **Inserção**: Insere um novo nó e realiza rotações para manter o balanceamento.
- **Remoção**: Remove um nó e ajusta a árvore para preservar o balanceamento.
- **Rotações**: São realizadas para corrigir desequilíbrios após inserções ou remoções. Podem ser rotações simples (à esquerda ou à direita) ou duplas (esquerda-direita ou direita-esquerda).

3.3 Aplicações

As Árvores AVL são amplamente utilizadas em sistemas que exigem acesso eficiente a dados, como:

- Bancos de dados: Para indexação e recuperação rápida de informações.
- Sistemas de arquivos: Para organizar diretórios e arquivos.
- Algoritmos de compressão: Para armazenar tabelas de frequência em estruturas balanceadas.
- Processamento de linguagem natural: Para tarefas como geração de paráfrases, onde o balanceamento da árvore melhora o desempenho em grandes conjuntos de dados.

3.4 Implementação da Geração de Paráfrases

A tarefa de geração de paráfrases consiste em substituir palavras de um texto original por sinônimos encontrados em um dicionário. A implementação utiliza uma Árvore AVL para armazenar o dicionário, onde:

- A chave é a palavra original.
- O valor associado é o sinônimo correspondente.

3.4.1 Estrutura de Dados

Cada nó da Árvore AVL armazena:

- **Chave:** Palavra do texto original.
- **Valor:** Palavra sinônima.
- **Filho Esquerdo e Direito:** Ponteiros para subárvores.
- **Altura:** Altura do nó na árvore, usada para calcular o fator de balanceamento.

Além disso, a Árvore AVL realiza rotações automáticas durante as inserções para garantir que a diferença entre as alturas das subárvores esquerda e direita seja no máximo 1. Isso assegura que a complexidade das operações seja mantida em $O(\log n)$, mesmo no pior caso.

3.4.2 Funcionamento do Algoritmo

O funcionamento do algoritmo para geração de paráfrases utilizando AVL é dividido em três etapas principais: carregamento do dicionário, processamento do texto e escrita do texto parafraseado. A seguir, cada etapa é descrita detalhadamente com exemplos de código.

1. **Carregamento do Dicionário:**
   - O dicionário é carregado a partir de um arquivo de texto onde cada linha contém uma palavra e seu sinônimo, separados por espaço. Um exemplo do formato do arquivo pode ser visto abaixo:

     ```
     palavra1 sinonimo1
     palavra2 sinonimo2
     ...
     ```

   - Cada entrada do dicionário é inserida na Árvore AVL utilizando a função `inserirAVL`.

     Durante a inserção, rotações são realizadas automaticamente sempre que necessário para manter o balanceamento da árvore. A lógica de inserção segue a ordem lexicográfica:
     - Palavras que vêm antes no alfabeto são posicionadas à esquerda.
     - Palavras que vêm depois no alfabeto são posicionadas à direita.

O trecho de Código 5 abaixo ilustra a função de inserção na Árvore AVL das palavras do dicionário:

```
NodoAVL* inserir_AVL(NodoAVL *raiz, char *chave, char *sinonimo) {
    if (raiz == NULL) {
        NodoAVL *novo = malloc(sizeof(NodoAVL));
        strcpy(novo->palavra, chave);
        strcpy(novo->sinonimo, sinonimo);
        novo->esq = novo->dir = NULL;
        novo->altura = 1;
        return novo;
    }
    if (strcmp(chave, raiz->palavra) < 0) {
        raiz->esq = inserir_AVL(raiz->esq, chave, sinonimo);
    } else if (strcmp(chave, raiz->palavra) > 0) {
        raiz->dir = inserir_AVL(raiz->dir, chave, sinonimo);
    }

    // Atualiza altura e realiza balanceamento
    raiz->altura = 1 + max(altura(raiz->esq), altura(raiz->dir));
    int balance = fator_balanceamento(raiz);

    // Realiza rotacoes conforme necessario
    if (balance > 1 && strcmp(chave, raiz->esq->palavra) < 0)
        return rotacao_direita(raiz);
    if (balance < -1 && strcmp(chave, raiz->dir->palavra) > 0)
        return rotacao_esquerda(raiz);
    if (balance > 1 && strcmp(chave, raiz->esq->palavra) > 0) {
        raiz->esq = rotacao_esquerda(raiz->esq);
        return rotacao_direita(raiz);
    }
    if (balance < -1 && strcmp(chave, raiz->dir->palavra) < 0) {
        raiz->dir = rotacao_direita(raiz->dir);
        return rotacao_esquerda(raiz);
    }

    return raiz;
}
```

Código 5: Código usado para inserir dados na árvore AVL

2. **Processamento do Texto:** O texto de entrada é processado palavra por palavra. Cada palavra é normalizada (removendo pontuações e convertendo para minúsculas) e buscada na Árvore AVL. Se um sinônimo for encontrado, ele substitui a palavra original; caso contrário, a palavra original é mantida.

   O Código 6 abaixo exemplifica a lógica de processamento:

```c
void parafrasear_AVL(FILE *entrada, FILE *saida, NodoAVL *dicionario,
                     int *comparacoes) {
    char palavra[100];
    int c;

    while ((c = fgetc(entrada)) != EOF) {
        if (isspace(c)) {
            fputc(c, saida); // Preserva espacos no texto
        } else if (ispunct(c)) {
            continue; // Ignora pontuacoes
        } else {
            ungetc(c, entrada);
            fscanf(entrada, "%99s", palavra);

            for (int i = 0; palavra[i]; i++) {
                palavra[i] = tolower((unsigned char)palavra[i]);
            }

            // Busca o sinonimo
            NodoAVL *sinonimo = consulta_AVL(dicionario, palavra,
                                             comparacoes);
            if (sinonimo) {
                fprintf(saida, "%s", sinonimo->sinonimo);
            } else {
                fprintf(saida, "%s", palavra);
            }
        }
    }
}
```

Código 6: Código usado para parafrasear na árvore AVL

   A normalização de palavras assegura que variações de maiúsculas e minúsculas não afetem a busca. Além disso, algumas pontuações específicas são removidas no texto de saída.

3. **Escrita do Texto Parafraseado:** Após processar cada palavra, o texto reformulado é salvo em um arquivo de saída. A função `fprintf` é utilizada para escrever tanto palavras substituídas quanto palavras originais, preservando espaços e formatação do texto.

O Código 7 também assegura que as estatísticas da Árvore AVL (como número de nós e altura) sejam registradas para análise de desempenho. Abaixo o Código 7

```c
void salvar_estatisticas_AVL(const char *
    arquivo_estatisticas,
                             const char *arquivo_entrada,
                             const char *arquivo_dicionario,
                             NodoAVL *raiz,
                             int comparacoes) {
    FILE *fp = fopen(arquivo_estatisticas, "w");
    if (!fp) {
        perror("Erro ao abrir o arquivo de estatisticas");
        return;
    }

    fprintf(fp, "======== ESTATISTICAS AVL ==========\n")
        ;
    fprintf(fp, "Arq Entrada: %s\n", arquivo_entrada);
    fprintf(fp, "Arq Dicionario: %s\n", arquivo_dicionario);
    fprintf(fp, "Numero de Nos: %d\n", contar_nodos_AVL(raiz
        ));
    fprintf(fp, "Altura da Arvore: %d\n", altura_AVL(raiz));
    fprintf(fp, "Numero de Rotacoes: %d\n",
        contar_rotacoes_AVL());
    fprintf(fp, "Numero de Comparacoes: %d\n", comparacoes);

    fclose(fp);
}
```

Código 7: Código usado para salvar estatística da árvore AVL

Essa abordagem modular permite que o algoritmo seja aplicado em diferentes contextos de geração de paráfrases com alta eficiência devido ao balanceamento automático da Árvore AVL.

4 Desempenho

4.1 Análise Teórica

As operações de busca e inserção na ABP possuem complexidade $O(h)$, onde h é a altura da árvore. Para uma árvore balanceada, $h \approx \log n$, sendo n o número de elementos. Assim, o desempenho da geração de paráfrases depende da eficiência da busca na árvore. Note também que por ser uma ABP, não é realizado qualquer rotação na árvore para aumentar seu desempenho, seja na busca, seja na inserção.

4.2 Resultados Experimentais

Nas próximas subseções, irá ser discutido o processo de paráfrase utilizando dois dicionários de tamanhos distintos para os dois tipos de árvores, juntamente com seus resultados.

4.2.1 Dicionário com 10 mil palavras por meio da árvore ABP

Os experimentos foram realizados utilizando um texto de entrada, como exemplo, o capítulo 1 do livro O Alienista, de Machado de Assis, e um dicionário com 10 mil palavras.

- **Número de Comparações:** 40.620 comparações foram realizadas durante a busca.
- **Número de Nós na Árvore:** 10.000.
- **Altura da Árvore:** 30.
- **Rotações:** 0 rotações foram realizadas.

4.2.2 Dicionário com 40 mil palavras por meio da árvore ABP

Os experimentos foram realizados utilizando um texto de entrada, como exemplo, o capítulo 1 do livro O Alienista, de Machado de Assis, e um dicionário com 40 mil palavras.

- **Número de Comparações:** 46.194 comparações foram realizadas durante a busca.
- **Número de Nós na Árvore:** 40.915.
- **Altura da Árvore:** 34.
- **Rotações:** 0 rotações foram realizadas.

4.2.3 Dicionário com 10 mil palavras por meio da árvore AVL

Os experimentos foram realizados utilizando um texto de entrada, como exemplo, o capítulo 1 do livro O Alienista, de Machado de Assis, e um dicionário com 10 mil palavras.

- **Número de Comparações:** 16.055 comparações foram realizadas durante a busca.
- **Número de Nós na Árvore:** 10.000.
- **Altura da Árvore:** 16 (graças ao balanceamento).
- **Número de Rotações:** 6.979 rotações realizadas durante as inserções.

4.2.4 Dicionário com 40 mil palavras por meio da árvore AVL

Os experimentos foram realizados utilizando um texto de entrada, como exemplo, o capítulo 1 do livro O Alienista, de Machado de Assis, e um dicionário com 40 mil palavras.

- **Número de Comparações:** 19.148 comparações foram realizadas durante a busca.
- **Número de Nós na Árvore:** 40.915.
- **Altura da Árvore:** 18 (graças ao balanceamento).
- **Número de Rotações:** 28.598 rotações realizadas durante as inserções.

4.2.5 Arquivos Utilizados

Os arquivos utilizados para o experimento podem ser baixados através dos links abaixo:

- **Arquivos de teste:** Baixar Aqui

4.3 Discussão Comparativa entre ABP e AVL

Os resultados mostram que o balanceamento automático da Árvore AVL reduz significativamente o número de comparações durante as buscas, especialmente em conjuntos grandes.

A tabela abaixo apresenta uma comparação direta entre ABP e AVL com base nos experimentos realizados com o dicionário de 10 mil palavras:

Métrica	ABP	AVL
Número de Nós	10.000	10.000
Altura da Árvore	30	16
Número de Comparações	40.620	16.055
Número de Rotações	0	6.979

Tabela 1: Comparação entre ABP e AVL na geração de paráfrases para o dicionário de 10 mil palavras.

Os resultados mostram que:

- A altura significativamente menor da Árvore AVL garante buscas mais rápidas, reduzindo o número total de comparações.
- O tempo total para processar o texto foi menor na AVL devido ao seu balanceamento eficiente.
- A ABP não realiza rotações, mas sua altura maior impacta negativamente no desempenho em conjuntos grandes.
- A AVL apresenta um custo adicional nas inserções devido às rotações realizadas para manter o balanceamento.

A tabela abaixo apresenta uma comparação direta entre ABP e AVL com base nos experimentos realizados com o dicionário de 40 mil palavras.

Métrica	ABP	AVL
Número de Nós	40.915	40.915
Altura da Árvore	34	18
Número de Comparações	46.194	19.148
Número de Rotações	0	28.598

Tabela 2: Comparação entre ABP e AVL para o dicionário de 40 mil palavras.

Com base nos dados apresentados na tabela acima, podemos tirar as seguintes conclusões sobre o desempenho das árvores ABP e AVL para o dicionário de 40 mil palavras:

- **Altura da Árvore:** A altura da ABP é significativamente maior (34) em comparação com a AVL (18). Isso reflete o impacto da falta de balanceamento na ABP, que cresce de forma desbalanceada dependendo da ordem das inserções. Na AVL, o balanceamento automático mantém uma altura menor, mesmo com um grande número de nós.

- **Número de Comparações:** A AVL realiza significativamente menos comparações (19.148) do que a ABP (46.194). Isso é consequência direta de sua altura menor, permitindo buscas mais rápidas e eficientes.
- **Número de Rotações:** A ABP não realiza rotações, o que simplifica a lógica de inserção. No entanto, isso resulta em uma estrutura menos eficiente. A AVL realiza 28.598 rotações para balancear a árvore durante as inserções, mas isso melhora muito o desempenho nas buscas.

4.3.1 Conclusões Gerais

- **Desempenho de Busca:** A AVL supera a ABP em desempenho de busca devido à sua menor altura. Isso é especialmente vantajoso para conjuntos grandes, como o dicionário de 40 mil palavras.
- **Custo das Inserções:** O custo adicional nas inserções causado pelas rotações da AVL é justificado pelo ganho expressivo de eficiência nas buscas subsequentes. Para aplicações com alta frequência de buscas e inserções moderadas, a AVL é a melhor escolha.
- **Uso Prático:** A ABP pode ser suficiente para pequenos conjuntos de dados ou cenários onde a ordem de inserção já produz uma árvore razoavelmente balanceada. No entanto, para grandes volumes de dados, como o dicionário de 40 mil palavras, a AVL é mais adequada devido ao seu desempenho superior em buscas e tempo total de execução.

Esses resultados reforçam que a AVL é a estrutura ideal para aplicações que exigem alta eficiência, especialmente em conjuntos grandes e operações frequentes de busca.

Com base nesses resultados, concluímos que a Árvore AVL é mais adequada para aplicações que exigem alto desempenho em conjuntos grandes e operações frequentes de busca.

4.4 Discussão dos Resultados

Como discutido anteriormente, o fato de a Árvore Binária de Pesquisa (ABP) não realizar rotações melhora seu desempenho em casos de conjuntos de dados pequenos. Entretanto, à medida que o volume de dados aumenta, o número de comparações cresce consideravelmente, o que sugere a necessidade de reconsiderar o uso da ABP em favor de uma árvore balanceada, como a AVL, para conjuntos de dados maiores.

5 Referências

Referências

[1] Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (3ª ed.). MIT Press.
Este livro fornece uma base teórica sólida sobre estruturas de dados, incluindo árvores binárias de pesquisa e algoritmos de busca.

[2] Sedgewick, R., & Wayne, K. (2011). *Algorithms* (4ª ed.). Addison-Wesley.
Explica detalhadamente estruturas de dados como árvores binárias, AVL e suas aplicações práticas.

[3] Goodrich, M. T., Tamassia, R., & Mount, D. M. (2004). *Data Structures and Algorithms in C++*. Wiley.
Excelente para compreender implementações práticas de árvores binárias e balanceadas em C/C++.

[4] GeeksforGeeks. *Binary Search Tree Data Structure*.
Disponível em: https://www.geeksforgeeks.org/binary-search-tree-data-structure/. Acesso em: 9 de janeiro de 2025.
Contém explicações práticas e implementações de árvores binárias de busca, além de exemplos em C.

[5] Programiz. *Binary Search Tree*.
Disponível em: https://www.programiz.com/dsa/binary-search-tree. Acesso em: 9 de janeiro de 2025.
Um guia simples para entender os fundamentos de árvores binárias de busca.

[6] Stack Overflow. *Perguntas e Respostas sobre Estruturas de Dados*.
Disponível em: https://stackoverflow.com/. Acesso em: 9 de janeiro de 2025.
Fórum utilizado para resolver dúvidas específicas sobre a implementação de algoritmos e estruturas de dados.

[7] Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (3ª ed.). MIT Press.
Este livro fornece uma base teórica sólida sobre estruturas de dados, incluindo árvores binárias de pesquisa e algoritmos de busca.

[8] Weiss, M. A. (1997). *Data Structures and Algorithm Analysis in C*. Addison-Wesley.
Apresenta uma análise detalhada de estruturas de dados balanceadas como AVL e suas vantagens em relação às ABPs.

[9] Knuth, D. E. (1998). *The Art of Computer Programming: Sorting and Searching* (2ª ed.). Addison-Wesley.
Um clássico que aborda algoritmos de busca e estruturas como árvores AVL com profundidade teórica.

[10] Kerridge, J., & Kerridge M. (2012). *Data Structures and Algorithms in C*. Springer.
Apresenta implementações detalhadas sobre árvores balanceadas como AVL e suas vantagens práticas.