



SERVIÇO PÚBLICO FEDERAL · MINISTÉRIO DA EDUCAÇÃO  
UNIVERSIDADE FEDERAL DE VIÇOSA · UFV  
CAMPUS FLORESTAL

## **Trabalho Prático - AEDS 2**

Estudo comparativo entre árvore PATRICIA e tabela HASH como estruturas para implementar arquivo invertido

BEATRIZ QUEIROZ CRUZ SILVA- 5890  
HEITOR PORTO JARDIM DE OLIVEIRA – 5895  
PÂMELA LÚCIA LARA DINIZ - 5898  
JÚLIO CESAR DE SOUZA OLIVEIRA - 5903  
OTÁVIO FRANCISCO SABINO TAVARES – 5912

## Sumário

1. Introdução .....	3
2. Metodologia .....	3
3. Detalhes técnicos.....	4
3.1 TAD Palavra .....	4
4. Resultados .....	5
4.1 PATRICIA.....	5
4.1.1 Análises e decisões para o funcionamento da estrutura.....	5
4.1.2 Implementação do índice invertido .....	6
4.1.3 Pesquisa de elementos .....	6
4.1.4 Impressão do índice invertido .....	7
4.2 HASH.....	7
4.2.1 Função de inserção com índice invertido .....	7
4.2.2 Impressão em ordem alfabética.....	7
4.3 Testes comparativos .....	8
4.3.1 Testes comparativos da Hash.....	8
4.3.2 Testes comparativos da PATRICIA .....	10
4.3.3 Conclusão comparativa.....	12
6. Conclusão .....	13
7. Referências.....	13

## 1. Introdução

O presente projeto visa a avaliação e comparação de desempenho das estruturas tabela Hash e árvore PATRICIA na construção de índices invertidos para uma máquina de busca a partir dos TCCs do curso de Ciência da Computação da UFV Campus Florestal. O programa, então, recebe N arquivos de entrada, insere as palavras dos POCs nas estruturas, constrói os índices invertidos referentes às palavras para cada uma das estruturas e os imprime, além de receber termos de busca por terminal, retornando os arquivos com relevância não nula, na ordem do mais relevante ao menos relevante, com o uso dos cálculos TF-IDF (Term Frequency – Inverse Document Frequency).

As principais funcionalidades foram feitas com base em Ziviani, 2011<sup>1</sup>, além de materiais externos como em Farrel, 2013<sup>2</sup> e Amaral, 2016<sup>3</sup>. Há, ainda, algumas implementações adjacentes baseadas em tutoriais e artigos do StackOverflow<sup>4</sup> e tutoriais do GeeksForGeeks<sup>5</sup> a serem referenciados no presente documento.

## 2. Metodologia

A divisão do trabalho se deu da seguinte maneira: inicialmente, cada integrante foi imputado de preparar uma porção dos POCs a serem usados. Já na implementação, foi estabelecido que três integrantes (Pâmela, Júlio e Otávio) trabalhassem no desenvolvimento da PATRICIA, e os demais (Beatriz e Heitor) desenvolvessem a tabela Hash. Para as implementações relacionadas ao menu (leitura de arquivo, tratamento de acentos, etc.) e para o desenvolvimento da lógica do TF-IDF, bem como a documentação de tais detalhes no README, foi escolhido um integrante para cada

---

<sup>1</sup> ZIVIANI, Nivio. Projeto de algoritmos com implementações em Pascal e C. 3ª Edição, São Paulo. Editora Cengage Learning, 1999. Implementações disponíveis em: <<https://www2.dcc.ufmg.br/livros/algoritmos/implementacoes.php>>. Último acesso em: 29 jun. 2025.

<sup>2</sup> PERL.COM. *Perl hash basics: create, update, loop, delete and sort*. 16 jun. 2013. Disponível em: <<https://www.perl.com/article/27/2013/6/16/Perl-hash-basics-create-update-loop-delete-and-sort/>>. Acesso em: 29 jun. 2025.

<sup>3</sup> AMARAL, Cleber Jorge. Programação I – Eng. Telecom 2016/1. IFSC Campus São José. Disponível em: <[https://wiki.sj.ifsc.edu.br/index.php/PRG29002\\_-\\_Programa%C3%A7%C3%A3o\\_I\\_-\\_Eng.Telecom\\_2016-1?utm\\_source=chatgpt.com#Acessando\\_arquivos\\_em\\_C](https://wiki.sj.ifsc.edu.br/index.php/PRG29002_-_Programa%C3%A7%C3%A3o_I_-_Eng.Telecom_2016-1?utm_source=chatgpt.com#Acessando_arquivos_em_C)>. Último acesso em: 29 jun. 2025.

<sup>4</sup> StackOverflow. Disponível em: <<https://stackoverflow.com/questions>> Último acesso em: 29 jun. 2025.

<sup>5</sup> GeeksForGeeks. Disponível em: <<https://www.geeksforgeeks.org/snprintf-c-library/>>. Acesso em: 10 jun. 2025.

(Otávio e Júlio, respectivamente). A comparação entre as estruturas ficou a cargo de dois integrantes (Pâmela e Beatriz), responsáveis pela análise da PATRICIA e da tabela Hash, respectivamente. O compartilhamento do projeto foi feito por meio do GitHub<sup>6</sup>.

### 3. Detalhes técnicos

#### 3.1 TAD Palavra

Para ambas as estruturas, foi utilizado o TAD Palavra como item. A struct do TAD foi nomeada como “TipoItemP”, e ele é responsável por armazenar a palavra em questão e o seu índice invertido. Dentro da estrutura há uma string contendo a palavra e uma lista encadeada contendo, em cada célula, a quantidade de vezes em que a palavra aparece no documento x, e o Id do documento.

As duas principais funções do TAD Palavra são a “imprime\_indice\_invertido” e a “QuantidadeTermosPorDoc”. A primeira tem a função de imprimir no terminal de forma organizada as informações da palavra. As informações são impressas no formato “palavra: <qtde, idDoc> | ... | <qtde, idDoc>”. Já a função “QuantidadeTermosPorDoc” retorna a quantidade de vezes em que a palavra está presente em um documento com ID passado por parâmetro. Essa função é importante para o TF-IDF (Term frequency – Inverse Document Frequency) para fazer os cálculos de relevância de cada arquivo.

```
typedef struct celula
{
    int idDoc;
    int qtde;
    struct celula * prox;
}Ccelula;

typedef struct TipoItemP
{
    char palavra[50] ;
    Ccelula * primeiro;
    int n_arquivos;
    int total_ocorrencias;
}TipoItemP;

void faz_palavra_vazia(TipoItemP * item, char * palavra);
void imprime_indice_invertido(TipoItemP * item);
void insere_palavra (TipoItemP * item, int idDoc);
int remove_palavra (TipoItemP * item, int idDoc);
int QuantidadeTermosPorDoc(TipoItemP item, int idDoc);
```

Figura 1 – Trecho do arquivo .h do TAD palavra

---

<sup>6</sup> Github. Disponível em: <[https://github.com/tavinescada/tp1\\_aeds2](https://github.com/tavinescada/tp1_aeds2)> Último acesso em: 30 jun. 2025.

## **4. Resultados**

### **4.1 PATRICIA**

A estrutura da árvore PATRICIA utilizada se baseou implementação de Nivio Ziviani, porém, ao invés de se trabalhar com uma árvore que armazena caracteres, a estrutura foi adaptada de modo a guardar palavras nos nós externos. Para isso, foram necessárias algumas análises a respeito do seu funcionamento para alcançar uma melhor solução para a distribuição das palavras. Sendo assim, nessa seção serão discutidos detalhes a respeito desse desenvolvimento.

#### **4.1.1 Análises e decisões para o funcionamento da estrutura**

O modelo inicial de árvore patricia com que trabalhamos armazenava no nó interno da árvore o índice referente ao caractere que se diferenciava entre dois termos representados por dígitos zero e um, essa representação binária permitia uma distribuição intuitiva uma vez que é possível direcionar caracteres “um” para a direita e “zero” para a esquerda por exemplo. A partir do momento em que se começa a trabalhar com palavras, ainda se mantém a característica de guardar em nós internos o índice de diferenciação, porém, é necessário considerar a existência de vinte e seis possibilidades de caracteres. Então, para que a estrutura funcionasse de maneira similar ao algoritmo estudado anteriormente, foi necessário adicionar mais um campo de comparação no nó interno da PATRICIA, que passa a armazenar também o primeiro caractere que difere entre duas palavras comparadas, e não apenas o índice.

Diante disso, a decisão a ser tomada passa a ser qual dos caracteres diferentes seriam guardados no nó interno da árvore, e, a partir dele, como seriam feitas as comparações para direcionar a inserção ou pesquisa de elementos. A opção adotada inicialmente foi de guardar no nó interno o maior caractere e a partir das comparações, direcionar aqueles maiores ou iguais a ele para a ramificação direita da árvore e os menores para a ramificação esquerda. Uma alternativa também considerada seria de armazenar o menor caractere no nó e direcionar aqueles menores ou iguais para a esquerda e maiores para a direita. A figura 1 retrata o trecho de código responsável por selecionar o caractere que será armazenado no nó interno durante a inserção, a partir da condição estabelecida de que seja o maior entre os dois já a figura 2 retrata a decisão de distribuição das palavras para a ramificação direita ou esquerda do nó inserido.

```

//Encontra o índice em que a palavra difere do nó atual
while (i<= strlen((const char*)palavra) && palavra[i] == NoAtual->NO.chave.palavra[i])
    i++;

if (i > strlen((const char*)palavra)){
    //Se entrar aqui, a palavra já existe na árvore, então só é incrementada a ocorrência
    insere_palavra(&NoAtual->NO.chave,idDoc);
    return (*NoRaiz);
}
else{
    caractere_dif=NoAtual->NO.chave.palavra[i];
    //garante que o caractere guardado no nó interno será sempre o maior
    //prefixos irão sempre para a esquerda pois \0 é menor que todos caracteres
    return InsereEntrePat(palavra, idDoc, NoRaiz, i, (palavra[i] > caractere_dif) ? palavra[i] : caractere_dif,comp_insercao_pat);
}

```

**Figura 2 – Trecho que define o caractere que será armazenado no nó interno.**

```

TipoArvore InsereEntrePat(char* palavra,int idDoc, TipoArvore *NoAtual, int i, char caractere_interno,int *comp_insercao_pat){
    TipoArvore NoExt;
    if (Eh_ExternoPat(*NoAtual) || i < (*NoAtual)->NO.NInterno.indice){
        NoExt = Cria_NO_ExternoPat(palavra,idDoc);

        if (palavra[i] >= caractere_interno){
            return (Cria_NO_InternoPat(i, NoAtual, &NoExt,caractere_interno)); //nova palavra a direita
        }
        else{
            return (Cria_NO_InternoPat(i, &NoExt, NoAtual,caractere_interno)); //nova palavra a esquerda
        }
    }
}

```

**Figura 3 – Trecho que define o padrão para a inserção de palavras na árvore.**

Essa adaptação garante que exista um padrão para que seja possível acessar qualquer elemento na estrutura, possibilitando a inserção e pesquisa de palavras.

#### 4.1.2 Implementação do índice invertido

Para realizar a montagem do índice invertido na PATRICIA, foi utilizado um TAD auxiliar “Palavra”, essa estrutura define um “TipoltemP” que carrega um campo para armazenamento da palavra, juntamente com uma lista encadeada em que cada célula representa uma ocorrência daquele termo em um documento distinto. Então, ao invés de inserir na árvore apenas a string referente às palavras, os nós externos irão receber uma estrutura de índice invertido associada à palavra inserida, que é atualizada durante a inserção dos elementos na árvore ao ser identificada a ocorrência ou não de repetições.

#### 4.1.3 Pesquisa de elementos

A pesquisa de elementos na estrutura PATRICIA acontece através da comparação com os caracteres dos nós internos, respeitando as definições de distribuição estabelecidas, que a busca de palavras com caracteres de comparação maiores ou iguais ao nó interno deve percorrer a ramificação direita do nó atual, caso contrário, seguirá pela ramificação esquerda, até que um nó externo seja encontrado.

#### 4.1.4 Impressão do índice invertido

Uma característica de árvores PATRICIA é que as palavras armazenadas na estrutura podem ser acessadas em ordem alfabética de maneira natural, ou seja, basta percorrer os nós de forma recursiva, realizando primeiramente chamadas para a ramificação esquerda e posteriormente visitando os nós à direita. Como cada nó externo possui, além da palavra, uma lista de índice invertido associado a ele, uma vez alcançado basta realizar a impressão do índice invertido referente ao termo encontrado por meio da chamada da função “imprime\_indice\_invertido” implementada pelo TAD “Palavra”. Isso garante que a listagem de índices será exibida em ordem alfabética no terminal.

## 4.2 HASH

Grande parte da estrutura e das funções básicas da tabela hash utilizadas neste trabalho foram adaptadas a partir do código clássico do professor Nivio Ziviani, da Universidade Federal de Minas Gerais (UFMG). Esse código fornece a base para a criação e manipulação de listas encadeadas em cada posição da tabela hash, bem como funções auxiliares como inicialização, inserção e busca de palavras. Foram realizadas duas modificações principais no código original.

O tamanho da tabela foi escolhido com base em um fator de carga de 0,7, resultando em um valor ideal aproximado de 218 posições. Assim, optou-se pelo número primo mais próximo, **223**, foram utilizados os primos **211** e **233** para fins de comparação nos testes.

### 4.2.1 Função de inserção com índice invertido

A função de inserção (Insere) foi alterada para permitir a construção e atualização de um índice invertido. Em vez de simplesmente inserir palavras na tabela hash, agora a inserção também registra a associação entre uma palavra e os documentos em que ela aparece, assim como a frequência de ocorrência em cada um. Para essa finalidade, utilizou-se um TAD adicional chamado palavra, responsável por armazenar a palavra, os identificadores dos documentos e suas respectivas contagens. A inserção verifica se a palavra já existe na tabela:

- Se já estiver presente, o índice invertido é atualizado com os dados do novo documento.
- Se ainda não estiver, a palavra é inserida e seu índice é iniciado.

### 4.2.2 Impressão em ordem alfabética

O código original de Ziviani realizava a impressão seguindo a ordem das posições da tabela hash. No entanto, como essa ordem é determinada pelos valores da função de hash, que por sua vez depende dos pesos aleatórios atribuídos, o resultado final não apresentava as palavras em ordem alfabética, como exigido pela

especificação do trabalho. Por esse motivo, foi necessário adaptar a lógica de impressão para garantir a ordenação correta do índice invertido.

Para atender a esse requisito, foi implementada uma nova lógica inspirada em um código retirado da internet, disponível no site da comunidade Perl (PERL.COM, 2013). A ideia foi adaptar a estratégia de ordenação de ponteiros para palavras usando a função `qsort` da linguagem C. Assim, percorremos toda a tabela hash, armazenamos os ponteiros para cada célula em um vetor, ordenamos esse vetor alfabeticamente com base nas palavras e, por fim, imprimimos os itens ordenados. Essa abordagem garante que a saída esteja em conformidade com o esperado, mesmo mantendo a estrutura original com listas encadeadas separadas por função de hash.

## 4.3 Testes comparativos

### 4.3.1 Testes comparativos da Hash

Com o objetivo de avaliar o desempenho da estrutura de tabela hash implementada neste trabalho, foram realizados testes comparativos voltados para a quantidade de **comparações** realizadas durante as operações de **inserção** e **pesquisa**. Foram considerados três tamanhos diferentes para a tabela: 211, 223 e 233 posições. Para manter a consistência dos testes, foi utilizada como base a **palavra "algoritmo"**. O número de comparações realizadas durante a inserção e a posterior pesquisa dessa palavra foi contabilizado em cada uma das versões da tabela. A seguir, são apresentados os resultados obtidos para cada configuração, com imagens geradas durante os testes e uma conclusão ao final.

```
10
11 // #define M 211//tamanho da tabela mudar de acordo com o Tp

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

--- Menu ---
1 - Ler o arquivo com os textos
2 - Construir os indices invertidos
3 - Exibir os indices invertidos
4 - Busca
0 - Fechar
2
Quantidade de comparacoes na insercao da Hash: 66306
Menu
```

Figura 4 - Inserção na Hash com M = 211.

```
Quantidade de comparacoes na pesquisa da Hash: 98
```

Figura 5 – Pesquisa na Hash com M = 211.



```
10
11 #define M 223//tamanho da tabela mudar de acordo
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
4 - Busca
0 - Fechar
2
Quantidade de comparacoes na insercao da Hash: 63398
```

**Figura 6 - Inserção na Hash com M = 223.**

Quantidade de comparacoes na pesquisa da Hash: 294

**Figura 7 – Pesquisa na Hash com M = 223.**

```
11 // #define M 233//tamanho da tabela mudar de acordo
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
3 - Exibir os indices invertidos
4 - Busca
0 - Fechar
2
Quantidade de comparacoes na insercao da Hash: 60863
```

**Figura 8 - Inserção na Hash com M = 233.**

Quantidade de comparacoes na pesquisa da Hash: 196

**Figura 9 – Pesquisa na Hash com M = 233.**

A partir dos testes com a palavra "algoritmo", foi possível observar que o número de comparações na pesquisa aumentou (de 98 a 294) conforme o tamanho da tabela hash aumentou de 211 a 223, e diminuiu (de 294 a 196) quando o tamanho foi de 223 para 233.

Além disso, os valores de inserção (66306, 63398, 60863) e até mesmo os de pesquisa variam não apenas com o tamanho da tabela, mas também com a geração aleatória dos pesos utilizados pela função de hash. Esses pesos afetam diretamente a posição para onde cada palavra é direcionada, influenciando o desempenho das operações. Por exemplo, no teste com a palavra "MyMobiConf", usando tabela de tamanho 223, a pesquisa exigiu 192 comparações em uma execução, mas saltou para 392 em outra, apenas por conta da mudança dos pesos.

Assim, conclui-se que o desempenho da tabela hash é volátil e depende de dois fatores principais: o tamanho da tabela e a configuração aleatória dos pesos, ambos influenciando diretamente o número de comparações realizadas nas operações de inserção e pesquisa.

### 4.3.2 Testes comparativos da PATRICIA

Uma vez implementado o algoritmo, foi possível realizar testes comparativos entre as duas alternativas estudadas, de armazenar o maior ou menor caractere nos nós internos, para que fosse encontrada a opção mais eficiente no que se refere à quantidade de verificações feitas pelas operações que envolvem a inserção e pesquisa na árvore. Então, a partir de uma mesma entrada de dados, as figuras 10 e 11 retratam a quantidade de comparações realizadas pela operação de inserção de palavras na PATRICIA.

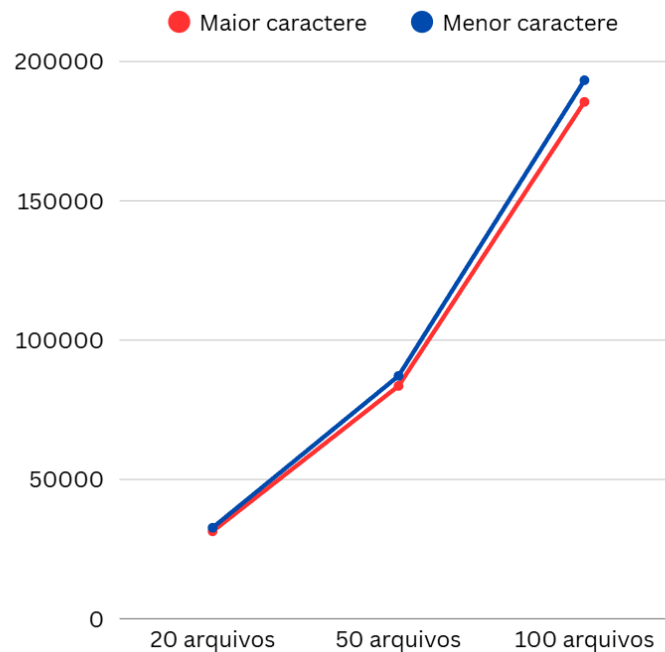
```
--- Menu ---  
1 - Ler o arquivo com os textos  
2 - Construir os indices invertidos  
3 - Exibir os indices invertidos  
4 - Busca  
0 - Fechar  
2  
Quantidade de comparacoes na insercao Patricia: 185515  
--- Menu ---
```

Figura 10 – Análise da inserção ao armazenar o maior caractere no nó interno.

```
--- Menu ---  
1 - Ler o arquivo com os textos  
2 - Construir os indices invertidos  
3 - Exibir os indices invertidos  
4 - Busca  
0 - Fechar  
2  
Quantidade de comparacoes na insercao Patricia: 193292  
--- Menu ---
```

Figura 11 – Análise da inserção ao armazenar o menor caractere no nó interno.

O gráfico representado na figura 12 retrata o comportamento das comparações, considerando o armazenamento de maior e menor caractere no nó interno e diferentes tamanhos de entrada. É possível observar que o comportamento é similar em menores entradas de dados, mas à medida que mais palavras são inseridas na estrutura, a quantidade de comparações com a utilização do menor caractere tende a crescer mais rapidamente.



**Figura 12 - Gráfico comparativo de eficiência da operação de inserção entre diferentes decisões de distribuição de elementos na árvore.**

A quantidade de comparações obtidas pela utilização de 100 arquivos indica uma média de 18 comparações por cada palavra inserida, uma vez que foram identificadas cerca de 10300 palavras como entrada do teste, isso se configura como um excelente desempenho para a estrutura.

Em seguida foi realizada a verificação de eficiência durante a operação de pesquisa de elementos na árvore, considerando as diferentes alternativas de distribuição já destacadas anteriormente neste documento e a utilização do cálculo de relevância dos termos.

Quantidade de comparacoes na pesquisa Patricia: 1442

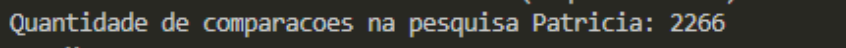
**Figura 13 - Comparações durante a pesquisa do termo “algoritmo”, considerando maiores caracteres nos nós interno.**

Quantidade de comparacoes na pesquisa Patricia: 1236

**Figura 14 - Comparações durante a pesquisa do termo “algoritmo”, considerando menores caracteres nos nós interno.**

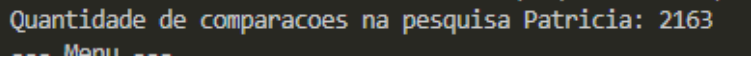
Quantidade de comparacoes na pesquisa Patricia: 2060

**Figura 15 - Comparações durante a pesquisa do termo “MyMobiConf”, considerando maiores caracteres nos nós interno.**



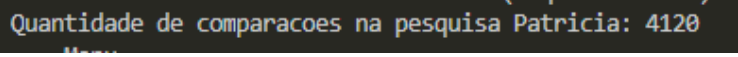
Quantidade de comparacoes na pesquisa Patricia: 2266

**Figura 16 - Comparações durante a pesquisa do termo “MyMobiConf”, considerando menores caracteres nos nós interno.**



Quantidade de comparacoes na pesquisa Patricia: 2163

**Figura 17 - Comparações durante a pesquisa do termo “Rede”, considerando maiores caracteres nos nós interno**



Quantidade de comparacoes na pesquisa Patricia: 4120

**Figura 18 - Comparações durante a pesquisa do termo “Rede”, considerando menores caracteres nos nós interno**

A partir dos testes realizados para a pesquisa de termos na árvore, observa-se que a busca da palavra “algoritmo” se saiu melhor quando o caractere dos nós internos se tratava do menor, com 1236 comparações realizadas contra 1442 verificações na árvore que armazena maiores caracteres nos nós. Já a busca do termo “MyMobiConf” desempenhou melhor a partir da configuração que armazena maiores caracteres nos nós internos, com 2060 comparações, contra 2266 referentes à outra configuração da árvore. Para a pesquisa do termo “Rede”, obteve-se a maior discrepância entre as duas configurações da árvore, com desempenho quase 100% melhor a partir do armazenamento das maiores letras nos nós internos.

Diante disso, pode-se concluir, apesar de algumas exceções, como foi observado pela pesquisa do termo “algoritmo”, que a pesquisa também desempenhou melhor a partir da escolha do maior caractere para os nós internos em um panorama geral.

Logo, a análise realizada com as duas operações na árvore PATRICIA indica que uma vez que se trabalha com essa estrutura, a alternativa de guardar o maior caractere que difere nos nós internos, distribuindo palavras maiores e iguais para a direita e menores para a esquerda, proporciona uma melhor eficiência quando se trabalha com uma maior quantidade de elementos. Isso acontece, pois, essa configuração realiza uma melhor distribuição das palavras aproveitando a propriedade de reciclagem de prefixos para otimização de acessos, característica principal da estrutura utilizada, melhorando o desempenho para a inserção e pesquisa de termos.

#### **4.3.3 Conclusão comparativa**

Diante as análises realizadas entre as duas estruturas, é possível concluir, por meio de um panorama geral que engloba a quantidade de comparações realizadas para as duas operações, que a tabela Hash teve um melhor desempenho

para lidar com o conjunto de dados utilizado e realizar pesquisas juntamente com o cálculo de relevância dos documentos.

## 6. Conclusão

As principais dificuldades no decorrer do desenvolvimento se encontraram em alguns momentos específicos, como no tratamento de acentos na entrada de dados, ao lidar com diferentes codificações de caracteres além da tabela ASCII. Além disso, a adaptação da estrutura original da PATRICIA para que fosse possível armazenar palavras se deparou com alguns dilemas até que uma conclusão eficiente fosse encontrada.

Por fim, por meio deste trabalho, pudemos compreender de forma prática como funciona a lógica por trás dos sistemas de busca na internet e como modificar as estruturas para corresponder às necessidades dessa aplicação.

## 7. Referências

ZIVIANI, Nivio. *Projeto de algoritmos com implementações em Pascal e C*. 3ª Edição, Minas Gerais. Editora Cengage Learning, 2011. Implementações disponíveis em: <<https://www2.dcc.ufmg.br/livros/algoritmos/implementacoes.php>>. Último acesso em: 29 jun. 2025.

MingGW. Disponível em: <<https://sourceforge.net/projects/mingw/>>. Último acesso em: 12 dez. 2024.

Github. Disponível em: <[https://github.com/tavinescada/tp1\\_aeds2](https://github.com/tavinescada/tp1_aeds2)> Último acesso em: 30 jun. 2025.

FARREL, David. *Perl hash basics: create, update, loop, delete and sort*. 16 jun. 2013. PERL.COM. Disponível em: <<https://www.perl.com/article/27/2013/6/16/Perl-hash-basics-create-update-loop-delete-and-sort/>>. Acesso em: 29 jun. 2025.

AMARAL, Cleber Jorge. *Programação I – Eng. Telecom 2016/1*. IFSC Campus São José. Disponível em: <[https://wiki.sj.ifsc.edu.br/index.php/PRG29002\\_-\\_Programa%C3%A7%C3%A3o\\_I\\_-\\_Eng.Telecom\\_2016-1?utm](https://wiki.sj.ifsc.edu.br/index.php/PRG29002_-_Programa%C3%A7%C3%A3o_I_-_Eng.Telecom_2016-1?utm)>. Último acesso em: 29 jun. 2025.

UTF-8-CHARTABLE. *Unicode UTF-8 character table – Latin Basic*. 2024. Disponível em: <<https://www.utf8-chartable.de/>>. Acesso em 28 jun. 2025.

LOCALIZELY. *CP850 character encoding*. 2024. Disponível em: <<https://localizely.com/character-encodings/cp850/>>. Acesso em: 28 jun. 2025.

TutorialsPoint. *File Handling in C*. Disponível em: <[https://www.tutorialspoint.com/cprogramming/c\\_file\\_io.htm](https://www.tutorialspoint.com/cprogramming/c_file_io.htm)>. Acesso em: 26 mai. 2025.

GeeksForGeeks. *strcspn()* in C. Disponível em:  
<<https://www.geeksforgeeks.org/strcspn-in-c/>>. Acesso em: 26 mai. 2025.

GeeksForGeeks. *snprintf()* in C. Disponível em:  
<<https://www.geeksforgeeks.org/snprintf-c-library/>>. Acesso em: 10 jun. 2025.

Microsoft Learn. *C Integer Constants*. Disponível em:  
<<https://learn.microsoft.com/en-us/cpp/c-language/c-integer-constants?view=msvc-170>>. Acesso em: 14 jun. 2025.

TULIO, Marco. *Mkdir function not working in C*. StackOverflow. Disponível em:  
<<https://stackoverflow.com/questions/25867348/mkdir-function-not-working-in-c>>.  
Acesso em: 19 jun. 2025.

Cppreference. *qsort*, *qsort\_s*. Disponível em:  
<<https://en.cppreference.com/w/c/algorithm/qsort>>. Acesso em: 23 jun. 2025.