Projeto II - INE5408
Indexação de Palavras com Estrutura de Trie

Nome do Aluno: Hayden Junior

Junho de 2025

Contents

1 Introdução

Este projeto tem como objetivo a construção de uma estrutura de dados do tipo *Trie* para armazenar e indexar palavras extraídas de arquivos com extensão `.dic`, que seguem o padrão do OpenOffice. A estrutura visa identificar prefixos em comum entre palavras e possibilitar a consulta eficiente desses prefixos. O projeto é desenvolvido como parte da disciplina de Estrutura de Dados e Algoritmos (INE5408), ministrada na Universidade Federal de Santa Catarina (UFSC).

2 Objetivos

- Desenvolver uma estrutura de dados eficiente para armazenar palavras com base em seus prefixos.
- Implementar operações de inserção e busca na *Trie*.
- Indexar as palavras de um dicionário padrão OpenOffice.
- Permitir a consulta de palavras com base em seus prefixos.
- Analisar o desempenho da estrutura.

3 Fundamentação Teórica

3.1 Estrutura de Dados Trie

Uma *Trie*, também conhecida como árvore de prefixos, é uma estrutura de dados especializada no armazenamento de strings, onde cada nó representa um caractere da palavra. Essa estrutura permite buscas, inserções e deleções em tempo proporcional ao tamanho da palavra, independentemente do número total de palavras armazenadas.

3.2 Vantagens

- Eficiência em buscas por prefixo.
- Utilização em algoritmos de sugestão/autocompletar.
- Estruturação hierárquica das palavras.

3.3 Arquivos .dic

Arquivos com extensão `.dic` são utilizados por processadores de texto, como o OpenOffice, para armazenar listas de palavras em um determinado idioma. Cada linha do arquivo contém uma palavra, podendo incluir marcações adicionais, como regras de flexão.

4 Metodologia

4.1 Leitura do Arquivo

O programa abre o arquivo `.dic` e lê cada linha, extraindo a palavra antes de qualquer marcador (como barras ou símbolos especiais). A linha é processada e, em seguida, a palavra é inserida na *Trie*.

4.2 Implementação da Trie

A estrutura `TrieNode` é composta por um vetor de ponteiros para os filhos (um para cada letra do alfabeto) e um marcador booleano indicando se aquele nó representa o fim de uma palavra válida.

Listing 1: Definição da estrutura TrieNode

```
typedef struct TrieNode {
    struct TrieNode* filhos[26];
    bool eh_fim_de_palavra;
} TrieNode;
```

4.3 Inserção na Trie

Para cada palavra, percorremos os caracteres e alocamos nós filhos caso ainda não existam. Ao final da palavra, o nó correspondente ao último caractere recebe a marcação de fim de palavra.

4.4 Busca por Prefixo

A busca por prefixo percorre a *Trie* até onde os caracteres do prefixo coincidirem. A partir desse ponto, todas as palavras derivadas daquele nó são listadas.

5 Desenvolvimento

5.1 Arquitetura do Programa

O programa é dividido em módulos para facilitar a organização:

- `trie.c` e `trie.h`: implementam a estrutura de dados.
- `main.c`: responsável pela leitura dos arquivos e interação com o usuário.

5.2 Desafios Encontrados

- ****Tratamento de palavras com acentuação: foi necessário filtrar ou adaptar a entrada.
- ****Gerenciamento de memória: a alocação dinâmica de nós exige cuidados para evitar vazamentos.
- ****Desempenho: otimizações foram necessárias para melhorar a velocidade de inserção e consulta.

6 Resultados

6.1 Testes Realizados

Foram utilizados os dois dicionários fornecidos para testar o desempenho e a correção do programa, além dos teste no VPL. Os testes incluíram:

- Inserção de todas as palavras do dicionário.
- Consulta de prefixos curtos (ex: “pre”) e longos (ex: “proprie”).

6.2 Análise

A *Trie* demonstrou excelente desempenho em buscas por prefixo, com tempos praticamente constantes para palavras já existentes. A estrutura também se mostrou eficiente no uso de memória para dicionários de médio porte.

7 Análise de Complexidade e Comparação com Algoritmos Tradicionais

A estrutura de dados *trie* foi escolhida neste projeto por sua eficiência na indexação e busca de prefixos em grandes volumes de palavras, como em arquivos de dicionários. A seguir, é apresentada uma análise da complexidade dos algoritmos utilizados, bem como uma comparação com o algoritmo de busca linear tradicional.

7.1 Busca Tradicional (Linear)

O algoritmo de busca tradicional, aplicado em listas ou vetores não ordenados, possui complexidade de tempo $\mathcal{O}(n)$, onde n representa o número total de palavras. Em cada busca, é necessário comparar a string de entrada com cada elemento da estrutura até que haja uma correspondência ou se esgotem os elementos.

Esse tipo de abordagem apresenta limitações de desempenho, especialmente em dicionários grandes, onde a busca por prefixos se torna ineficiente.

7.2 Busca com Trie

A *trie* é uma árvore de prefixos na qual cada nó representa um caractere e caminhos formam palavras completas. A complexidade da operação de busca na trie é $\mathcal{O}(m)$, onde m é o comprimento da palavra (ou prefixo) buscado, independentemente da quantidade de palavras armazenadas.

Como cada caractere leva a um próximo nó da árvore, o tempo de busca depende apenas da quantidade de letras do prefixo e não da quantidade total de palavras na trie. Esta característica torna a *trie* ideal para buscas por prefixos, operações de autocompletar e verificação de existência de palavras.

7.3 Comparação

Table 1: Comparação entre os algoritmos de busca

Algoritmo	**Complexidade da Busca**	**Dependência do Tamanho do Dicionário**
Busca Linear	$\mathcal{O}(n)$	Sim
Busca com Trie	$\mathcal{O}(m)$	Não

Observa-se que, enquanto a busca linear se torna cada vez mais custosa conforme o número de palavras cresce, a busca na *trie* mantém seu desempenho estável mesmo com grandes volumes de dados. Essa vantagem é especialmente importante em

aplicações que exigem respostas rápidas, como corretores ortográficos, sistemas de sugestão e mecanismos de busca com autocompletar.

8 Conclusão

O uso da *Trie* se mostrou uma solução eficaz para o problema de indexação de palavras por prefixo. A estrutura é simples de implementar e oferece ótimo desempenho, sendo adequada para aplicações como corretores ortográficos e sistemas de sugestão automática.

9 Referências

- CORMEN, Thomas H.; LEISERSON, Charles E.; RIVEST, Ronald L.; STEIN, Clifford. *Algoritmos: Teoria e Prática*. 3. ed. Rio de Janeiro: Elsevier, 2012.
- SEDGEWICK, Robert; WAYNE, Kevin. *Algorithms*. 4th Edition. Addison-Wesley, 2011.
- OpenOffice Dictionary Format. Disponível em: `https://wiki.openoffice.org/wiki/Dictionaries`. Acesso em: junho de 2025.
- **Material didático da disciplina INE5408 - Estrutura de Dados e Algoritmos (UFSC)**, incluindo slides e exemplos fornecidos em aula.