UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO - UFERSA
CAMPUS MULTIDISCIPLINAR DE PAU DOS FERROS - CMPF
BACHARELADO INTERDISCIPLINAR EM TECNOLOGIA DA INFORMAÇÃO
DISCIPLINA: LABORATÓRIO DE ALGORITMOS E ESTRUTURA DE DADOS II

Mateus Gomes Pinheiro - Redator
Marcelo Henrique De Lima Marques - Apresentador
Alisson Lima Ricarte - Git Master
Francisco Daniel Costa de Souza - Codificador

Relatório Acadêmico: Projeto de um Dicionário em C utilizando Tabela Hash

Prof. Kennedy Reurison Lopes

Pau dos Ferros 2025

RESUMO

Este relatório apresenta o desenvolvimento de um dicionário digital em linguagem C utilizando a estrutura de dados conhecida como tabela hash. O projeto foi realizado como atividade prática da disciplina de Estrutura de Dados II e teve como objetivo principal implementar um sistema capaz de inserir, buscar, remover, salvar e carregar palavras de forma eficiente. Para isso, utilizou-se uma tabela hash com encadeamento separado como método de resolução de colisões. A estrutura adotada consistiu em um vetor de listas encadeadas, onde cada índice é calculado por uma função hash aplicada à palavra. A manipulação de arquivos foi implementada para garantir a persistência dos dados, permitindo a recuperação do conteúdo do dicionário em execuções futuras do programa. Os testes realizados comprovaram a eficácia da estrutura, apresentando bom desempenho nas operações e correta distribuição das palavras, mesmo em situações com colisões. A remoção de elementos e o carregamento/salvamento em arquivos funcionaram conforme o esperado. O projeto permitiu a aplicação prática de conceitos fundamentais da disciplina, como alocação dinâmica, uso de ponteiros, listas encadeadas e funções de hash. Conclui-se que a estrutura de tabela hash é eficiente para a construção de sistemas de busca e armazenamento de dados textuais, e que a implementação em linguagem C proporciona uma compreensão profunda dos mecanismos internos dessas estruturas.

Palavras-chave: tabela hash, linguagem C, dicionário digital, estrutura de dados, listas encadeadas.

SUMÁRIO

INTRODUÇÃO

O avanço das tecnologias digitais tem impulsionado a criação de estruturas de dados cada vez mais eficientes para lidar com o crescente volume de informações. Nesse contexto, as tabelas hash se destacam como uma solução eficaz para armazenamento, busca e gerenciamento rápido de dados. Este projeto propõe o desenvolvimento de um dicionário digital em linguagem C, utilizando a estrutura de tabela hash como mecanismo principal para a organização e recuperação de palavras.

O trabalho foi concebido como atividade prática da disciplina de Estrutura de Dados II (ED2), com o objetivo de consolidar o aprendizado teórico por meio de uma aplicação concreta. O sistema desenvolvido permite inserir, buscar, remover, salvar e carregar palavras de um dicionário, promovendo a manipulação dinâmica de dados e sua persistência. Dessa forma, o projeto não apenas reforça o domínio sobre estruturas de dados avançadas, como também estimula a prática da programação modular e o uso de ponteiros e listas encadeadas..

OBJETIVOS

Objetivo geral

- Desenvolver um dicionário digital utilizando a linguagem C e a estrutura de tabela hash, com suporte a operações básicas de manipulação e persistência de dados.

- Implementar uma tabela hash com tratamento de colisões adequado para armazenar palavras;

- Criar funções eficientes para inserção, busca e remoção de palavras na estrutura;
- Implementar mecanismos de leitura e escrita em arquivos, permitindo a salvaguarda e recuperação do dicionário entre execuções do programa;
- Aplicar os conceitos de alocação dinâmica de memória e manipulação de listas encadeadas;
- Consolidar o conhecimento prático sobre estruturas de dados e algoritmos, com ênfase em eficiência e organização do código.

FUNDAMENTAÇÃO TEÓRICA

A estrutura de tabela hash é uma das mais utilizadas na ciência da computação quando se deseja obter eficiência nas operações de busca, inserção e remoção de dados. Ela funciona a partir de uma função chamada função hash, que transforma uma chave (neste caso, uma palavra) em um índice de um vetor, onde os dados são armazenados. A principal vantagem desse tipo de estrutura é seu tempo médio constante de acesso (O(1)), independentemente da quantidade de elementos armazenados.

No entanto, diferentes chaves podem gerar o mesmo índice — fenômeno conhecido como colisão. Para lidar com isso, são adotadas técnicas de resolução, sendo a mais comum e utilizada neste projeto o encadeamento separado (*chaining*), no qual cada posição da tabela aponta para uma lista encadeada de elementos que colidiram naquele índice.

O uso de tabelas hash é amplamente adotado em sistemas reais, como compiladores, bancos de dados, sistemas de cache e mecanismos de busca, o que reforça a importância de seu estudo e domínio em cursos de ciência da computação e engenharia de software.

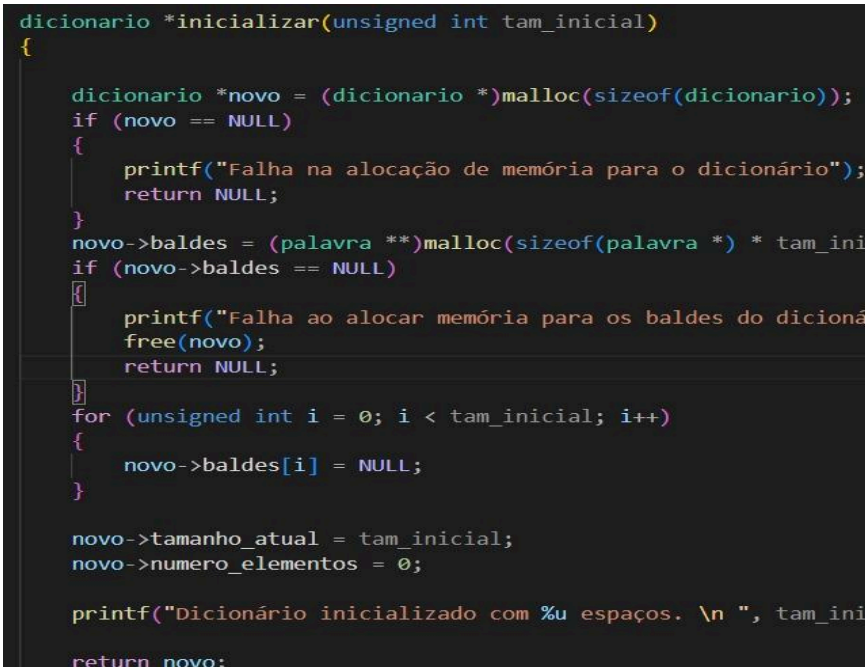
METODOLOGIA

O projeto foi desenvolvido utilizando a linguagem C, com ênfase em estruturas dinâmicas e eficientes para manipulação de dados textuais. A estrutura central do dicionário é uma **tabela hash com encadeamento separado**, onde as palavras e seus significados são armazenados em listas ligadas que ocupam os “baldes” (buckets) da tabela.

Estrutura da Tabela Hash

A tabela hash é implementada como um vetor de ponteiros para estruturas do tipo **palavra**, que por sua vez contêm listas de significados. Cada posição do vetor representa um bucket, o qual pode armazenar múltiplas palavras em caso de colisões. A função **inicializar** é responsável por alocar dinamicamente a estrutura do dicionário, criar o vetor de buckets com o tamanho especificado e inicializá-lo com valores nulos. Além disso, os campos **tamanho atual** e **número de elementos** são configurados para controle interno.

Essa estrutura permite que diferentes palavras com índices hash iguais compartilhem o mesmo bucket, sendo organizadas como uma lista encadeada.



```c
dicionario *inicializar(unsigned int tam_inicial)
{

    dicionario *novo = (dicionario *)malloc(sizeof(dicionario));
    if (novo == NULL)
    {
        printf("Falha na alocação de memória para o dicionário");
        return NULL;
    }
    novo->baldes = (palavra **)malloc(sizeof(palavra *) * tam_ini
    if (novo->baldes == NULL)
    {
        printf("Falha ao alocar memória para os baldes do dicioná
        free(novo);
        return NULL;
    }
    for (unsigned int i = 0; i < tam_inicial; i++)
    {
        novo->baldes[i] = NULL;
    }

    novo->tamanho_atual = tam_inicial;
    novo->numero_elementos = 0;

    printf("Dicionário inicializado com %u espaços. \n ", tam_ini

    return novo;
```

Função Hash

A função **calcular hash** é a responsável por transformar uma palavra (string) em um índice numérico válido dentro dos limites da tabela. Para isso, foi utilizado o algoritmo **djb 2**, reconhecido por sua boa distribuição de chaves. O processo inicia com um valor fixo (5381) e, para cada caractere da string, realiza operações de deslocamento e adição que alteram progressivamente o valor da hash. O resultado final é ajustado com o operador módulo (%) com o tamanho da tabela para garantir que o índice esteja dentro do intervalo permitido.

Esse mecanismo garante uma distribuição eficiente das palavras nos buckets, reduzindo a ocorrência de colisões excessivas.

Exemplo de função calcular hash em linguagem c:

```c
unsigned int calcular_hash(const char *palavra, unsigned int tamanho_tabela)
{
    unsigned long hash = 5381;
    int c;

    while ((c = *palavra++))
    {

        hash = ((hash << 5) + hash) + c;
    }

    return hash % tamanho_tabela;
}
```

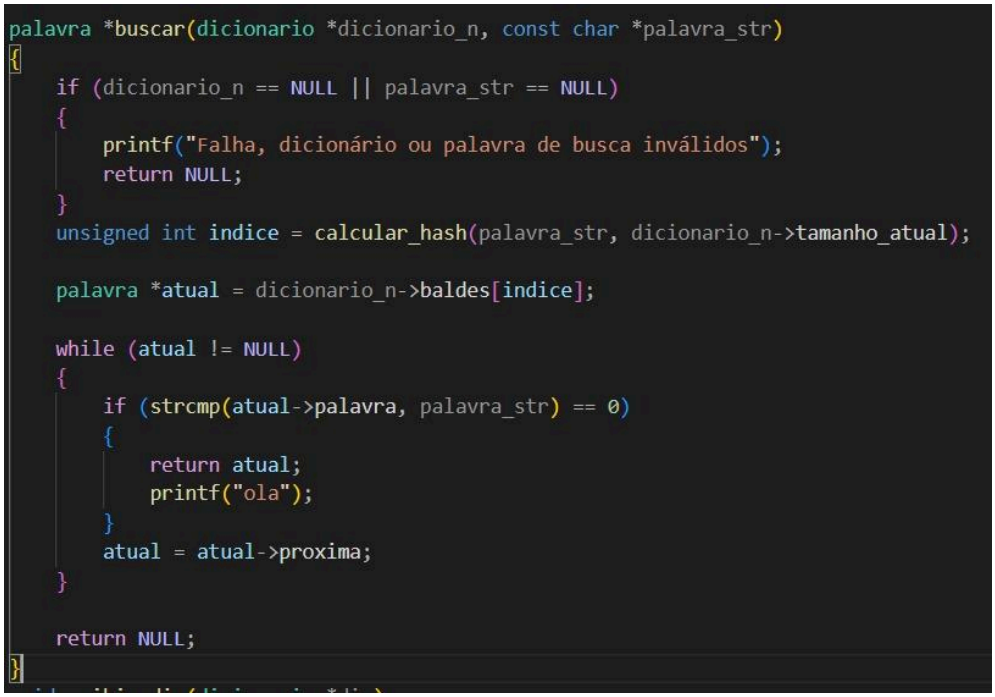
Operações Implementadas

Diversas operações foram implementadas para permitir o uso pleno do dicionário:

- **Inserção de palavras:**
  A função **inserir palavra** calcula o índice do bucket usando a função hash e verifica se a palavra já existe na lista encadeada. Caso já exista, apenas adicione um novo significado à palavra. Se não existir, cria um novo nó com a palavra e insere o primeiro significado. Essa inserção é feita no início da lista, o que garante um tempo constante (O(1)) para a operação.

- **Busca de palavras:**
  A função **buscar** localiza uma palavra específica no dicionário. Ela calcula o índice do bucket, percorre a lista encadeada e retorna um ponteiro para o nó da palavra se encontrada. Caso contrário, retorna **NULL**. A eficiência da busca é garantida pela limitação do espaço de busca a apenas um bucket.

Exemplo de função de buscar palavras em linguagem c:



```c
palavra *buscar(dicionario *dicionario_n, const char *palavra_str)
{
    if (dicionario_n == NULL || palavra_str == NULL)
    {
        printf("Falha, dicionário ou palavra de busca inválidos");
        return NULL;
    }
    unsigned int indice = calcular_hash(palavra_str, dicionario_n->tamanho_atual);

    palavra *atual = dicionario_n->baldes[indice];

    while (atual != NULL)
    {
        if (strcmp(atual->palavra, palavra_str) == 0)
        {
            return atual;
            printf("ola");
        }
        atual = atual->proxima;
    }

    return NULL;
}
```

- **Remoção de palavras:**

  A função **remover palavra** percorre o bucket correspondente até encontrar a palavra, ajustando os ponteiros da lista encadeada para excluí-la. Todos os significados associados são liberados da memória, assim como o próprio nó da palavra.

- **Remoção de significados:**

  Com a função **remover significado**, é possível apagar um significado específico de uma palavra existente. Ela localiza a palavra e remove apenas o nó do significado desejado, mantendo a palavra e os demais significados intactos.

- **Adição de novos significados:**

  Caso a palavra já exista, a função permite adicionar significados adicionais através da mesma lógica de inserção, vinculando-os à lista de significados da palavra existente.

- **Exibição do dicionário:**
  A função **exibir dic** percorre todos os buckets da tabela, exibindo as palavras encontradas e todos os seus significados, servindo como uma listagem geral do conteúdo atual do dicionário.

- **Validação de strings:**
  A função **caracteres permitidos** assegura que as palavras inseridas contenham apenas caracteres válidos (letras, espaço, vírgula, hífen, apóstrofo).

- **Ajuste de strings:**
  A função de **aparar espaços** remove espaços em branco no início e no final de uma string, garantindo a integridade da chave.

- **Liberação de memória:**
  A função **liberar dicionário** percorre toda a tabela hash e libera cada pedaço de memória alocado, tanto para as palavras quanto para os significados, evitando vazamentos de memória ao final do programa.

Essas funções são ativadas por meio de um menu interativo na função principal (main), com **cases** específicos que permitem inserir, buscar, remover palavras ou significados, listar todas as palavras e encerrar o programa de forma segura.

CONCLUSÃO

O desenvolvimento de um dicionário digital utilizando a estrutura de tabela hash em linguagem C proporcionou uma experiência prática essencial para a consolidação dos conceitos abordados na disciplina de Estrutura de Dados II. A estrutura proposta demonstrou ser eficiente na realização de operações fundamentais como inserção, busca, remoção e persistência de dados.

O uso de encadeamento separado para o tratamento de colisões garantiu estabilidade e desempenho nas buscas, mesmo em situações com múltiplas palavras alocadas em um mesmo bucket. A implementação modular e o uso de ponteiros, listas encadeadas e manipulação de arquivos reforçaram habilidades importantes no contexto da programação em baixo nível.

Como resultado, o sistema desenvolvido demonstrou funcionalidade completa e coerente com os objetivos propostos. Para trabalhos futuros, recomenda-se a expansão do projeto com recursos como interface gráfica, armazenamento em arquivos binários e suporte a definições mais complexas por palavra, ampliando seu potencial como ferramenta didática e técnica.

REFERÊNCIAS

CORMEN, Thomas H. et al. *Algoritmos: teoria e prática*. 3. ed. Rio de Janeiro: Elsevier, 2012.

ZIVIANI, Nivio. *Projeto de algoritmos com implementações em Pascal e C*. Cengage Learning, 2015.

Material didático da disciplina de Estrutura de Dados II – *Prof. Kennedy Reurison Lopes*

MANBER, Udi. *Introduction to Algorithms: A Creative Approach*. Reading, MA: Addison-Wesley, 1989.

ABNT. NBR 14724:2023 – Informações e documentação – Trabalhos acadêmicos – Apresentação. Associação Brasileira de Normas Técnicas, 2023.

ANEXOS

Anexo A – Código-fonte principal do dicionário

Contém as funções centrais da implementação do dicionário digital, incluindo:

- Inicialização da estrutura;
- Função hash (djb2);
- Inserção, busca e remoção de palavras;
- Liberação de memória.