UNIVERSIDADE FEDERAL DE ALFENAS

ALGORITMOS E ESTRUTURA DE DADOS II

ISABELA MAGESTE DE ANDRADE

JOAQUIM PEDRO DO NASCIMENTO MOREIRA DE JESUS

VICTÓRIA DE ALMEIDA TAMBASCO

ALFENAS

2025/2

RESUMO

Este trabalho apresenta a implementação de um algoritmo para a identificação de palíndromos utilizando estruturas de dados em linguagem C. A aplicação foi desenvolvida como parte das atividades da disciplina de Algoritmos e Estruturas de Dados II, do curso de Ciência da Computação da Universidade Federal de Alfenas (UNIFAL). O algoritmo busca a leitura de expressões textuais e identificação da presença de palíndromos. Os testes demonstraram a eficácia do método proposto na identificação correta de palíndromos, mesmo em expressões com acentuação e mistura de casos. O desenvolvimento seguiu boas práticas de programação e alocação dinâmica de memória, respeitando as restrições estabelecidas pela disciplina, tendo como resultado um código funcional e que atende ao problema proposto.

LISTA DE FIGURAS

SUMÁRIO

1. INTRODUÇÃO

Segundo o site TodaMatéria, “Palíndromo, do grego *palin* (novo) e *dromo* (percurso), é toda palavra ou frase que pode ser lida de trás para frente e que, independente da direção, mantém o seu sentido” (TODAMATÉRIA, 2023). O reconhecimento de palíndromos é um problema clássico da computação, pois envolve técnicas de manipulação de dados, normalização de texto e uso eficiente de estruturas de dados.

Dessa forma, este trabalho, buscou resolver esse problema explorando o uso de estruturas dinâmicas, especificamente, listas encadeadas duplas, previamente implementadas em aula, sem o uso de bibliotecas ou funções nativas de manipulação de strings da linguagem C. O algoritmo proposto realiza a leitura de expressões textuais a partir de um arquivo, realiza a normalização dos caracteres (removendo acentos, espaços e símbolos), e verifica se a sequência resultante é simétrica, utilizando uma lista duplamente encadeada para permitir comparações entre os caracteres das extremidades da expressão.

O projeto tem como principal objetivo colocar em prática os conhecimentos adquiridos ao longo da disciplina de Algoritmos e Estruturas de Dados II, reforçando a importância do uso correto e eficiente de estruturas de dados na solução de problemas computacionais reais. Além disso, buscou-se estimular o trabalho em equipe, a organização do código, o controle de versões e a documentação adequada do projeto.

Este trabalho foi desenvolvido de forma colaborativa utilizando o editor Visual Studio Code (VS Code) e a plataforma GitHub, permitindo a integração e sincronização entre os membros do grupo ao longo do desenvolvimento. O relatório foi organizado em seções que abrangem os objetivos do projeto, a descrição das decisões técnicas, a implementação do algoritmo e, por fim, a análise dos resultados obtidos.

2. OBJETIVOS

2.1 OBJETIVOS GERAIS :

- Elaborar e implementar um algoritmo que identifique se uma expressão (palavra, frase ou número) é um palíndromo utilizando as estruturas ensinadas na disciplina de AEDS II.

2.2 OBJETIVOS ESPECÍFICOS:

- Desenvolver funções para normalizar as expressões, removendo espaços, pontuação, acentos e convertendo letras para minúsculas.

- Implementar as estruturas de dados fila e pilha em linguagem C para armazenar os caracteres das expressões.

- Comparar os caracteres da fila e da pilha para verificar a simetria da expressão.

- Exibir o resultado das comparações e determinar se as condições de simetria foram concluídas.

3. DESENVOLVIMENTO

O desenvolvimento do trabalho foi dividido em duas etapas principais. Inicialmente, o grupo realizou discussões para analisar diferentes estratégias para resolver o problema proposto, avaliando suas vantagens e limitações em relação aos requisitos estabelecidos. Posteriormente, iniciou-se a fase de implementação, na qual as ideias selecionadas foram codificadas, testadas e refinadas para garantir a correta funcionalidade do algoritmo. Para o controle de versão e gerenciamento do código-fonte, utilizamos a plataforma GitHub, que permitiu a colaboração eficiente entre os integrantes do grupo e o histórico das modificações realizadas.

3.1 IDEALIZAÇÃO

Durante a fase de idealização, três abordagens principais foram consideradas para a detecção de palíndromos utilizando as estruturas de dados trabalhadas em sala:

- **Uso de lista encadeada simples:** a ideia inicial consistia em criar uma lista encadeada simples para armazenar os caracteres da expressão, realizando um loop para comparar os elementos nas pontas da lista (início e fim) e avançar até o meio. Essa abordagem foi descartada, pois a estrutura de lista simples não permite o acesso direto ao elemento final sem percorrer toda a lista, o que tornaria a comparação ineficiente e o algoritmo incorreto para o problema.

- **Lista encadeada simples combinada com vetor:** uma alternativa considerada foi criar a lista encadeada simples e, em seguida, copiar seus elementos para um vetor, invertendo a ordem deste vetor para comparação com a lista original. Entretanto, essa solução foi rejeitada, pois envolve o uso de um vetor — uma estrutura de dados básica que não foi contemplada no escopo do trabalho — e não respeita a exigência de utilizar exclusivamente estruturas complexas desenvolvidas em sala de aula (listas, pilhas e filas).

- **Lista duplamente encadeada:** por fim, optamos por implementar uma lista duplamente encadeada, que permite a navegação bidirecional através dos ponteiros para os elementos anteriores e posteriores. Essa estrutura possibilita a comparação direta dos caracteres nas extremidades da lista, avançando simultaneamente pelos dois sentidos até encontrar o ponto médio. Dessa forma, foi possível manter o controle eficiente sobre os elementos a serem comparados sem a necessidade de estruturas auxiliares ou cópias intermediárias. Esta abordagem atendeu de maneira satisfatória aos requisitos do trabalho e foi a estratégia implementada.

3.2 IMPLEMENTAÇÃO

Para a implementação da ideia selecionada, foi desenvolvido um programa em linguagem C estruturado em dois arquivos principais: um arquivo de cabeçalho, “palindromo.h” contendo as definições de estruturas e funções, e o arquivo principal, “palindromo.c”, responsável pela leitura do arquivo, chamada das funções e controle de fluxo da aplicação. O algoritmo foi dividido em três etapas principais: normalização da entrada, armazenamento em lista duplamente encadeada e verificação da simetria para identificação do palíndromo.

3.2.1 Estruturas de dados

Dentro do projeto, foram usadas as seguintes estruturas de dados:

- **Struct**: Foi definida uma *struct* chamada *word*, que representa os nós de uma lista duplamente encadeada, conforme exposto na FIGURA 1. Cada nó armazena um caractere de uma palavra, além de ponteiros para o nó anterior e o próximo. Essa estrutura permitiu a manipulação eficiente dos dados necessários para a análise de simetria das palavras.

FIGURA 1 - STRUCT WORD

```c
typedef struct word
{
    char caractere;
    struct word *prox;
    struct word *ant;
} palindromo;
```

fonte: autoria própria

- **Lista Duplamente Encadeada**: Trata-se de uma estrutura de dados linear composta por uma sequência de elementos, onde cada elemento (nó) contém um dado e dois ponteiros: um para o próximo elemento e outro para o elemento anterior da lista. Essa característica permite percorrer a lista em ambas as direções — do início ao fim e do fim ao início — facilitando

operações como inserção, exclusão e busca em qualquer ponto da lista. Segundo o site *Programação Dinâmica* (2023), “uma lista duplamente encadeada é uma coleção de nós onde cada nó possui dois ponteiros, um para o nó seguinte e outro para o nó anterior, permitindo assim uma navegação bidirecional eficiente” (PROGRAMAÇÃO DINÂMICA, 2023).

No contexto do projeto, a lista duplamente encadeada foi implementada com base na *struct word*. Essa estrutura possui um ponteiro para o próximo nó e outro para o nó anterior, o que facilitou o percurso nos dois sentidos da palavra (do início ao fim e do fim ao início). Essa característica foi essencial para verificar se uma palavra é simétrica (palíndroma) ou não, dentro do projeto.

- **Ponteiros:** Os ponteiros foram utilizados em vários contextos ao longo do código para manipular de maneira dinâmica as demais estruturas de dados. Em particular, para manipular os nós da lista de forma dinâmica. Como exemplo, os ponteiros “inicio” e “fim” mantêm o controle das extremidades da lista durante a verificação. O ponteiro “inicio” aponta para o nó sentinela, enquanto “fim” aponta para o último nó válido inserido, como ilustrado na FIGURA 2.

FIGURA 2 - PONTEIROS

```
palindromo *esquerda = inicio->prox;
palindromo *direita = fim;
```

fonte: autoria própria

3.2.2 Algoritmo

Nesta seção, apresenta-se o funcionamento do algoritmo desenvolvido para a verificação de palíndromos a partir da leitura de um arquivo texto. O programa realiza a leitura linha a linha, processando cada expressão para identificar se ela é um palíndromo, ou seja, se pode ser lida da mesma forma de trás para frente. Para isso, o código utiliza uma lista duplamente encadeada para armazenar e comparar os caracteres das palavras de forma eficiente.

3.2.2.1 Leitura e Normalização dos Caracteres

O algoritmo começa lendo os caracteres de cada linha do arquivo, ignorando espaços, pontuações e outros símbolos não alfabéticos. Para lidar com caracteres acentuados, comuns na língua portuguesa, foi implementada uma função que converte esses caracteres para sua forma simples, sem acento. Por exemplo, letras como ‘á’ ou ‘É’ são transformadas em ‘A’ e ‘E’, respectivamente.

Esse processo é realizado na função ler_linha(), que lê um caractere por vez e o normaliza, conforme a FIGURA 3.

FIGURA 3- FUNÇÃO ler_linha()

```
bool ler_linha(FILE *arquivo, palindromo **fim)
{
    int byte1, byte2;

    while ((byte1 = fgetc(arquivo)) != '\n' && byte1 != EOF)
    {
        char caractere_normalizado;
        int bytes_consumidos;

        if ((byte1 & 0xE0) == 0xC0)
        {
            byte2 = fgetc(arquivo);
            if (byte2 == EOF)
                break;

            caractere_normalizado = converter_acento((unsigned char)byte1, (unsigned char)byte2, &bytes_consumidos);
        }
        else
        {
            caractere_normalizado = converter_acento((unsigned char)byte1, 0, &bytes_consumidos);
        }
        if (caractere_normalizado > 90)
        {
            caractere_normalizado -= 32;
        }

        if (caractere_normalizado != '\0')
        {
            palindromo *novo = (palindromo *)malloc(sizeof(palindromo));
            novo->caractere = caractere_normalizado;
            novo->prox = NULL;
            novo->ant = *fim;

            (*fim)->prox = novo;
            *fim = novo;
        }
    }

    return true;
}
```

fonte: autoria própria

Aqui, a função converter_acento() recebe os bytes que compõem o caractere e retorna a versão “sem acento” e em maiúscula. Caso o caractere não seja uma letra, ele é ignorado.

3.2.2.2 Construção da Lista Duplamente Encadeada

Cada caractere válido é armazenado em um nó de uma lista duplamente encadeada. Esta lista permite que o algoritmo percorra os caracteres tanto do início para o fim quanto do fim para o início, o que é fundamental para a comparação que verifica a simetria da palavra.

O código insere um novo nó na lista a cada caractere lido, mantendo os ponteiros para os nós anterior e próximo atualizados, conforme a FIGURA 4.

FIGURA 4 - LISTA DUPLAMENTE ENCADEADA

```
palindromo *novo = (palindromo *)malloc(sizeof(palindromo));
novo->caractere = caractere_normalizado;
novo->prox = NULL;
novo->ant = *fim;

(*fim)->prox = novo;
*fim = novo;
```

fonte: autoria própria

Dessa forma, a lista representa a palavra ou frase normalizada, pronta para ser avaliada.

3.2.2.3 Verificação da Simetria dos Caracteres

Com a lista pronta, o algoritmo compara os caracteres das extremidades — o primeiro e o último — avançando os ponteiros em direção ao centro da lista. Se em algum momento os caracteres forem diferentes, a expressão não é um palíndromo.

O trecho responsável pela verificação é indicado na FIGURA 5.

FIGURA 5 - VERIFICAÇÃO DE PALÍNDROMO

```
palindromo *esquerda = inicio->prox;
palindromo *direita = fim;

while (esquerda != NULL && direita != NULL &&
       esquerda != direita && esquerda->ant != direita)
{
    if (esquerda->caractere != direita->caractere)
    {
        return false;
    }
    esquerda = esquerda->prox;
    direita = direita->ant;
}
return true;
```

fonte: autoria própria

Esse laço percorre os caracteres simultaneamente dos dois lados, garantindo que a palavra seja simétrica.

3.2.2.4 Liberação da Memória

Após a verificação, os nós alocados para a lista são liberados da memória para evitar vazamentos, utilizando a função liberar_lista():

FIGURA 6 - VERIFICAÇÃO DE PALÍNDROMO

```c
void liberar_lista(palindromo *inicio)
{
    palindromo *atual = inicio;
    while (atual != NULL)
    {
        palindromo *proximo = atual->prox;
        free(atual);
        atual = proximo;
    }
}
```

fonte: autoria própria

Assim, a memória utilizada para armazenar os caracteres de cada linha é limpa antes da próxima iteração.

3.2.2.5 Fluxo Principal

No programa principal (main()), o arquivo é aberto e, para cada linha, é feita a leitura e normalização dos caracteres, a construção da lista, a verificação do palíndromo e a liberação da lista. O resultado é exibido como 1 (sim) ou 0 (não):

FIGURA 7 - VERIFICAÇÃO DE PALÍNDROMO

```
while (!feof(arquivo))
{
    sucesso = ler_linha(arquivo, &fim);

    if (!sucesso)
    {
        printf("Erro ao ler a linha do arquivo.\n");
        fclose(arquivo);
        return 1;
    }

    if (verifica_palindromo(inicio, fim))
    {
        printf("1\n");
    }
    else
    {
        printf("0\n");
    }
    liberar_lista(inicio->prox);
    inicio->prox = NULL;
    fim = inicio;
}
```

fonte: autoria própria

Este ciclo se repete até o final do arquivo, garantindo que todas as linhas sejam processadas de forma independente.

Com essa implementação, o algoritmo consegue identificar palíndromos de maneira eficiente, mesmo quando as expressões contêm letras acentuadas ou estão em maiúsculas ou minúsculas, devido à normalização aplicada.

CONCLUSÃO

O desenvolvimento deste projeto permitiu a consolidação dos conhecimentos adquiridos ao longo da disciplina de Algoritmos e Estruturas de Dados II, por meio da aplicação prática de estruturas dinâmicas, como listas duplamente encadeadas, na resolução de um problema clássico da computação: a identificação de palíndromos.

A implementação do algoritmo foi cuidadosamente elaborada para garantir o correto processamento de expressões textuais, considerando a diversidade de caracteres da língua portuguesa, como acentos e letras maiúsculas/minúsculas. A etapa de normalização foi essencial para assegurar uma análise precisa, removendo inconsistências e permitindo a comparação direta entre os caracteres.

A utilização de uma lista duplamente encadeada revelou-se adequada e eficiente para este tipo de tarefa, pois viabilizou a navegação bidirecional necessária à verificação da simetria das expressões, sem a necessidade de estruturas auxiliares. Além disso, a manipulação dinâmica da memória e o controle preciso dos ponteiros demonstraram a importância de uma implementação cuidadosa e consciente ao trabalhar com estruturas complexas em linguagem C.

Por fim, o projeto atendeu aos objetivos propostos, promovendo o desenvolvimento da lógica algorítmica, a prática da programação em baixo nível e o uso adequado de estruturas de dados não lineares. Além disso, a organização do trabalho em equipe e o uso de ferramentas como o GitHub para versionamento reforçaram boas práticas no desenvolvimento de software acadêmico.

4. BIBLIOGRAFIA

PROGRAMAÇÃO DINÂMICA. **Estrutura de dados: lista duplamente encadeada.** Disponível em: https://www.programacaodinamica.com.br/estrutura-de-dados-lista-duplamente-encadeada/

TODAMATÉRIA. ***Palíndromo – significado, origem e exemplos*.** Disponível em: https://www.todamateria.com.br/palindromo/