Relatório dissertativo da Atividade Avaliativa 1 (22/08/2025)

Objetivo do documento: comparar o desempenho dos códigos produzidos, a partir do exercício “Entrega 01”, do dia 22/08/2025, com as devidas especificações, sendo cobrados dois eixos de resolução: uma pela IA e outra, “caseira” (modo recursivo e iterativo). No caso, será discutido o desempenho final observado (notação “Big O”), além dos pontos nos quais foram contrastantes entre o desenvolvimento de cada caminho, para a resolução da situação problema em si.

Observação: a função “int main()” utilizada em todos os programas a serem abordados tem leves diferenças, mas a variável “TAMANHO” (igual a 100, na implementação desse trabalho/relatório) e as bibliotecas importadas (“stdio.h”, “string.h” e “ctype.h”) são as mesmas.

```c
int main() {
    char palavra[TAMANHO];

    // Leitura de strings atC) EOF
    while (fgets(palavra, sizeof(palavra), stdin)) {
        palavra[strcspn(palavra, "\n")] = '\0'; // Remove o '\n' adicionado por fgets

        if (palindromoVerificadorIterativo(palavra))
            printf("Sim\n");

        else
            printf("Nao\n");
    }

    return 0;
}
```

Função main() utilizada pelo programador

```c
int main() {
    char palavra[TAMANHO]; // Palavra de até 100 caracteres, incluindo espaços.
    while(fgets(palavra, sizeof(palavra), stdin)){
        if (palindromoVerificador(palavra)) // Uso de função do tipo bool para a condicional.
            printf("Sim\n");

        else
            printf("Não\n");
    }

    return 0;
}
```

Função main() utilizada pelo programador

É possível reparar que a IA foi mais perspicaz em realizar uma leitura mais segura dos dados, já que trata melhor para os casos de “NULL” e de “EOF”.

Versão Iterativa)

Primeiramente, temos o código gerado pelo Sabiá-3.1, da Maritaca AI:

```c
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <stdbool.h>

#define TAMANHO 100

// Função para verificar se a string é um palíndromo de forma iterativa
bool palindromoVerificadorIterativo(char *palavra) {
    int len = strlen(palavra);
    int esquerda = 0, direita = len - 1;

    while (esquerda < direita) {
        // Ignora caracteres não alfanuméricos à esquerda
        while (esquerda < direita && !isalnum((unsigned char)palavra[esquerda])) {
            esquerda++;
        }

        // Ignora caracteres não alfanuméricos à direita
        while (esquerda < direita && !isalnum((unsigned char)palavra[direita])) {
            direita--;
        }

        // Compara os caracteres alfanuméricos em posições simétricas
        if (tolower((unsigned char)palavra[esquerda]) != tolower((unsigned char)palavra[direita])) {
            return false; // Não é um palíndromo
        }

        // Move os índices em direção ao centro
        esquerda++;
        direita--;
    }

    return true; // É um palíndromo
}
```

Em seguida, temos a versão feita pelo programador:

```c
bool palindromoVerificador(char *palavra) { // Função de checagem das palavras palíndromas.
    char resultado[TAMANHO];
    int j = 0;

    for (int i = 0; i < strlen(palavra); i++) { // Trecho de conversão da palavra original para letras minúsculas apenas em letras do alfabeto.
        if (isalnum(palavra[i])) {
            resultado[j] = tolower(palavra[i]);
            j++;
        }
    }

    for (int i = 0; i < j/2; i++) { // Loop de checagem final do suposto palíndromo.
        if (resultado[i] != resultado[j - 1 - i])
            return false;
    }

    return true;
}
```

Discussão técnica: o modelo desenvolvido pela IA apresentou uma alternativa a qual não envolveu a criação de uma “sub-string” da frase original declarada pelo usuário na main(), priorizando, assim, o uso de variáveis inteiras de índice do vetor de caracteres, associadas com mais iterações condicionais (estruturas “while” encadeadas e aninhadas), sem alterar o período lexical fornecido ao programa. Houve um foco maior no percorrimento do vetor, assim, é notável uma maior quantidade de comparações no total (3 situações condicionais), sob uma perspectiva exata do total de operações (comparações, percorrimento de vetor, declaração de variáveis) que ocorreriam durante a execução do programa. Nota-se, portanto, uma natureza de problema escalável pela perspectiva de execução de tempo O(n), uma vez que as comparações não são retrocedidas ao se percorrer o vetor.

Já o código feito pelo programador, deu ênfase em um viés de uso de string temporária, utilizada para checagem, logo após a conversão dela em uma frase sequencial de caracteres aglomerados de letras minúsculas. A comparação, a partir da string interna à função, utilizou o princípio de “two pointers” (dois ponteiros distintos percorrendo as extremidades do vetor para facilitar a comparação). Percebe-se uma guinada notável para o

uso ostensivo de variável vetorial temporária, a fim de simplificar o total de comparações necessárias para a determinação do argumento booleano (verdadeiro ou falso) demandado na main(). Logo, a complexidade de tempo, também, é de natureza O(n).

Caso	Status	Tempo de CPU
Caso 1	Correto	0.0012 s
Caso 2	Correto	0.0011 s
Caso 3	Correto	0.0012 s
Caso 4	Correto	0.0011 s
Caso 5	Correto	0.0011 s

Tempo de execução do método iterativo (versão “caseira”)

Caso	Status	Tempo de CPU
Caso 1	Correto	0.0012 s
Caso 2	Correto	0.0011 s
Caso 3	Correto	0.0012 s
Caso 4	Correto	0.0011 s
Caso 5	Correto	0.0011 s

Tempo de execução do método iterativo (versão da IA)

Versão Recursiva)

A versão desenvolvida, em questão, é esta:

```
bool palindromoVerificarRecursivo(char *palindromoVerificador, int meio, int deslocamento); // Checagem de palíndromos via recursão.
bool palindromoVerificador(char *palavra); // Checagem de palíndromos via iteração.

bool palindromoVerificador(char *palavra) { // Função iterativa de checagem das palavras palíndromas.
    char resultado[TAMANHO];
    int j = 0;

    for (int i = 0; i < strlen(palavra); i++) { // Trecho de conversão da palavra original para letras minúsculas apenas em letras do alfabeto.
        if (isalnum(palavra[i])) {
            resultado[j] = tolower(palavra[i]); // Conversão de quaisquer caracteres para letras minúsculas
            j++;
        }
    }

    return palindromoVerificarRecursivo(resultado, j, 0); // Retorno final condicionado pela função recursiva de checagem.
}

bool palindromoVerificarRecursivo(char *palindromoRecursivo, int fim, int inicio) { // Função recursiva de checagem das palavras palíndromas.
    if (inicio >= fim) // Se, durante as chamadas "encapsuladas" da recursão, a igualdade for verdadeira, então é palíndromo.
        return true;

    if (palindromoRecursivo[inicio] != palindromoRecursivo[fim - 1]) // Comparação de caracteres
        return false;

    return palindromoVerificarRecursivo(palindromoRecursivo, fim - 1, inicio + 1); // Chamada recursiva efetiva.
}
```

Trecho modificado para a versão recursiva de resposta do problema

Discussão técnica: é visível o gasto de memória o qual o método recursivo proporciona, já que depende de chamadas em cadeia de uma função qualquer sobre si mesma, sendo de natureza O(n) para o problema apresentado, já que dependeria exatamente do “comprimento” da frase a ser analisada. Dado que o problema não apresenta quaisquer cenários mais elaborados/complexos, não se

percebe, por isso, muita diferença se comparado com o método iterativo, tanto no tempo de execução, quanto na notação “Big O” utilizada para qualificar analiticamente o problema em si, apesar de se ter, naturalmente, mais operações realizadas (no caso, as comparações das estruturas condicionais “if”).

Caso	Status	Tempo de CPU
Caso 1	Correto	0.0012 s
Caso 2	Correto	0.0011 s
Caso 3	Correto	0.0011 s
Caso 4	Correto	0.0011 s
Caso 5	Correto	0.0012 s

Tempo de execução do método recursivo

Discussões Comparativas)

Observou-se que a IA não somente foi mais atenta no tratamento de dados incertos recebidos pela função de verificação de palíndromos, notoriamente visto no uso do “casting” do caractere lido na comparação para “unsigned char”, mas também tratou com mais segurança a entrada de dados na main(), já que considerou os casos de erro do scanf() e o fim da leitura dos dados de forma indeterminada (“NULL” e “EOF”). Além disso, a IA mostrou, apesar de ter sido mais verborrágica para efetuar as operações necessárias, um claro raciocínio sobre o problema solicitado, como também foi o caso da versão “caseira”, dado que priorizou em simplificar a string original (frase declarada pelo usuário) para uma forma de letras minúsculas sequenciais para, em seguida, aplicar a comparação de forma mais direta usando a estratégia “two pointers”. Esta versão, inclusive, teve como prioridade o raciocínio direto, sem se declarar muitas variáveis e com o objetivo de simplificar as notações das comparações efetivas e, logicamente, a quantia total delas.

A diferença de desempenho de tempo de execução do código entre as versões apresentadas durante o relatório se mostra mínima, já que ela se dá em uma natureza O(1), por envolver condicionais “if” a menos e pela ausência de um trecho adicional de iteração (“for” ou “while”), aplicável para cada eixo de desenvolvimento explanado previamente. Dessa maneira, o comportamento linear não é alterado e, para o contexto, não há uma notória diferença no total de comparações, já que se percebe diferença no tempo de execução entre os métodos iterativos, assim como no método recursivo.

Por fim, o método iterativo seria melhor que o método recursivo, porque este método gasta muito mais recurso de memória da máquina para garantir a cadeia de chamadas das funções declaradas no programa. Não há, entre as metodologias apresentadas/discutidas devidamente, diferenças significativas quanto ao tempo de execução do código, pois há poucos fatores O(1) como influência direta.