Relatório dissertativo da Atividade Avaliativa 3 (12/09/2025)

Objetivo do documento: comparar o desempenho dos códigos produzidos, a partir do exercício “Entrega 03 – pt 01”, do dia 12/09/2025, com as devidas especificações, sendo cobrados dois eixos de resolução: uma de forma iterativa e a outra de forma recursiva. 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.

Versão Iterativa)

Temos a seguinte versão:

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct {
    int *tamanhos;
    int tamanho;
    int capacidade;
} Grupo;

void adicionarTamanho(Grupo *grupo, int tamanho) {
    if (grupo->tamanho >= grupo->capacidade) {
        grupo->capacidade += 10;
        grupo->tamanhos = realloc(grupo->tamanhos, grupo->capacidade*sizeof(int));
    }

    grupo->tamanhos[grupo->tamanho] = tamanho;
    grupo->tamanho++;
}

int contarCaracteres(char *nome) {
    int tamanho = 0;

    for (int i = 0; nome[i] != '\0'; i++) {
        if (nome[i] != ' ')
            tamanho++;
    }
```

```c
int contarCaracteres(char *nome) {
    int tamanho = 0;

    for (int i = 0; nome[i] != '\0'; i++) {
        if (nome[i] != ' ')
            tamanho++;
    }

    return tamanho;
}

void imprimirGrupo(char *tipo, Grupo grupo) {
    printf("%s - [", tipo);

    for (int i = 0; i < grupo.tamanho; i++) {
        printf("%d", grupo.tamanhos[i]);

        if (i < grupo.tamanho - 1)
            printf(", ");
    }

    printf("]");
}
```

```c
int main() {
    Grupo usp = {NULL, 0, 0};
    Grupo externa = {NULL, 0, 0};
    char buffer[256];

    while (fgets(buffer, sizeof(buffer), stdin)) {
        char nome[256], tipo[256];

        sscanf(buffer, "%[^-]-%s", nome, tipo); // Regex para fazer função split()

        int tamanho = contarCaracteres(nome);

        if (strcmp(tipo, "usp") == 0)
            adicionarTamanho(&usp, tamanho);

        else if (strcmp(tipo, "externa") == 0)
            adicionarTamanho(&externa, tamanho);
    }

    imprimirGrupo("USP", usp);
    printf("\n");
    imprimirGrupo("Externa", externa);
    printf("\n");

    return 0;
}
```

Discussão técnica: A implementação utiliza uma ideia componentizada em Structs, a fim de facilitar a contagem de caracteres de cada nome pertencente a

um grupo específico (usp ou “externa”), o total de elementos capazes de serem armazenados no “vetor” de números de caracteres e a referência, justamente, do array de inteiros, sendo que as variáveis usadas para esses objetivos, respectivamente, são: tamanho, capacidade e tamanhos. Primeiramente, o usuário declara, em um loop “while()”, os nomes dos participantes e o grupo em que pertencem, ambos os campos separados por “- “, como entrada padronizada para o tratamento de dados da situação problema exposta no documento. Em seguida, aplica-se uma lógica de contagem de caracteres simples, desconsiderando os espaços em branco de nomes compostos declarados pelo usuário, gerando um resultado inteiro que servirá como argumento para inserir tal resultado no array de caracteres contados, para cada grupo participante. Por fim, ocorrerá a impressão das contagens e do “vetor” mencionado, a partir de uma função de impressão genérica (“imprimirGrupo()”).

É possível notar que o desempenho tem duas perspectivas: tempo e memória. Para a primeira medida, percebe-se um desempenho $O(n^2)$, já que, conta com uma iteração “while()”, sob ritmo O(n), estruturas “if()” relativas O(1), exclusivamente, ao total de inputs colocados pelo usuário, além do fato da função de “adicionarTamanho()” conter uma lógica de alocação dinâmica. Compondo o desempenho de todas as estruturas, tem-se o $O(n^2)$, onde $O(n^2) = O(n*n + 1)$, considerando o fato da impressão, também, ser O(n) [$O(n^2) + O(n) = O(n^2)$]. Já para a segunda medida, observa-se que é de natureza O(n), já que a struct Grupo armazena n inteiros em cada array associado, garantindo um consumo de memória devidamente linear.

Resultado dos Casos:

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

Resultados e temporização da resolução iterativa nos casos de teste do runcodes

Versão Recursiva)

A versão desenvolvida, em questão, é esta:

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct {
    int *tamanhos;
    int tamanho;
    int capacidade;
} Grupo;

void adicionarTamanho(Grupo *grupo, int tamanho) {
    if (grupo->tamanho >= grupo->capacidade) {
        grupo->capacidade += 10;
        grupo->tamanhos = realloc(grupo->tamanhos, grupo->capacidade*sizeof(int));
    }

    grupo->tamanhos[grupo->tamanho] = tamanho;
    grupo->tamanho++;
}

int contarCaracteres(char *nome, int tamanho) {
    if (nome[tamanho] == '\0')
        return 0;

    return (nome[tamanho] != ' ') + contarCaracteres(nome, tamanho + 1);
}
```

```c
void imprimirGrupo(char *tipo, Grupo grupo) {
    printf("%s - [", tipo);

    for (int i = 0; i < grupo.tamanho; i++) {
        printf("%d", grupo.tamanhos[i]);

        if (i < grupo.tamanho - 1)
            printf(", ");
    }

    printf("]");
}
```

```c
int main() {
    Grupo usp = {NULL, 0, 0};
    Grupo externa = {NULL, 0, 0};
    char buffer[256];

    while (fgets(buffer, sizeof(buffer), stdin)) {
        char nome[256], tipo[256];

        sscanf(buffer, "%[^-]-%s", nome, tipo); // Regex para fazer função split()

        int tamanho = contarCaracteres(nome, 0);

        if (strcmp(tipo, "usp") == 0)
            adicionarTamanho(&usp, tamanho);

        else if (strcmp(tipo, "externa") == 0)
            adicionarTamanho(&externa, tamanho);
    }

    imprimirGrupo("USP", usp);
    printf("\n");
    imprimirGrupo("Externa", externa);
    printf("\n");

    return 0;
}
```

Discussão técnica: Essa versão teve apenas como diferença, em relação a sua versão iterativa, a forma pela qual a contagem foi disposta, saindo do padrão tradicional via “for()” para chamadas sucessivas da função “contarCaracteres()” em si mesma. Dessa forma, a complexidade de tempo ainda é mantida em $O(n^2)$, assim como a de memória em O(n), uma vez que apenas houve um aumento no total de operações elementares O(1) na função de contagem de caracteres exposta, por conta dos “call-backs”.

Resultado dos Casos:

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

Resultados e temporização da resolução recursiva nos casos de teste do runcodes

Discussões Comparativas)

Analisando a forma pela qual o código foi desenvolvido, assim como as sutis diferenças no tempo de execução da CPU para os 5 casos de teste padronizados outrora pelo runcodes, podemos concluir que, para esse caso, a versão iterativa é melhor do que a recursiva, porque não conta com operações extra de referenciação sucessiva da função de contagem “contarCaracteres()”, apesar de ter a mesma estrutura de tratamento do tamanho do array de contagem dos nomes ditado por uma lógica alicerceada pelo uso da struct Grupo. Poderia ser possível reduzir, para ambos os casos, a complexidade de tempo, a partir da

realocação de posições de forma exponencial, o que garantiria uma amortização do gasto computacional para lidar com o incremento linear de espaço na memória necessário para lidar com a capacidade de dados limite estipulada por Grupo.

Logo, a forma iterativa evita o uso extra da pilha de memória e permite que o compilador aplique otimizações de laço mais agressivas, sendo, então, uma forma mais preferível para a compilação. Entretanto, a versão recursiva, apesar de ser mais legível, gera um risco maior de “overhead” por chamada de função, podendo estourar a pilha nos casos de nomes muito longos.