Universidade Federal de São João del-Rei
Departamento de Ciência da Computação
Estrutura de Dados
Professor Guilherme de Castro Pena

Instruções para entrega:

- Entregue a lista apenas no formato *.pdf* com o nome ***Y_listaX.pdf***, onde **X é o número da lista** e **Y é o número da sua matrícula**. Não serão aceitos outros formatos.
- Inclua nome e matrícula, e mantenha a resolução dos exercícios **ordenada** e **legível**.
- Códigos completos (com int main), em texto, sem executáveis.

  Para cada um, apresente uma imagem da tela de saída do seu programa.
- Após a data de entrega, a nota da entrega é 0.
- Em caso de dúvidas, procurem os monitores. Haverá um monitor após as aulas de laboratório para tirar dúvidas sobre a lista.

Lista de Exercícios 2
Deque, Fila de Prioridades, Matrizes, Árvores

Data máxima de entrega: 02/06/2025 - 11:59h
(Entrega: pelo SIGAA, na sua turma de Estrutura de Dados.)

1 Deque

1.1 Utilizando um deque de caracteres, escreva uma função para verificar se uma palavra é um palíndromo. Teste sua função.

1.2 Considere um Deque Duplamente Encadeado que armazena números inteiros. Faça uma função que receba um DDE e exclua todos os números negativos. A ordem dos outros elementos não deve ser alterada. Teste sua função.

1.3 Crie uma função que imprima os elementos de um deque na ordem inversa (do fim até o início). Faça para os TADs Deque Sequencial Estático e Deque Duplamente Encadeado. Teste sua função.

1.4 Considere um deque capaz de armazenar caracteres. Implemente uma função que retorne se as duas metades do deque são simétricas. Teste sua função. Exemplos:

- A B A A B A -> verdadeiro
- A B C X Y A -> falso
- X Y Z Y X -> verdadeiro

1.5 Explique com suas palavras como implementar uma fila utilizando um deque e como implementar uma pilha utilizando um deque.

2 Fila de Prioridades

2.1 Implemente uma função que receba duas filas de prioridade (F1 e F2) e faça a união de ambas. O resultado da união deve ser colocado em F1. A fila F2 deve ficar vazia. Faça a função para ambos os tipos de fila: Dinâmica e Estática;

2.2 Reimplemente o **TAD: Fila de Prioridades com Heap Binária** mas para funcionar como um **heap de mínimo**, ou seja, a menor prioridade fica à frente, o item a ser removido será o de menor prioridade agora. Crie uma *Main.c* com um menu (do..while) com as seguintes opções, chamando as operações:

- 1 - Criar Fila;
- 2 - Inserir um item pela prioridade;
- 3 - Ver o início da Fila.
- 4 - Remover um item;
- 5 - Imprimir a Fila;
- 6 - Mostrar o tamanho da Fila;
- 7 - Destruir a Fila;
- 8 - Sair;

3 Matrizes

3.1 Inclua para o TAD de matriz sequencial, três operações matemáticas da seguinte forma, para cada operação, receba duas matrizes, e se possível (testar dimensões), crie uma terceira matriz que conterá o resultado da referida operação:

- Soma entre duas matrizes, elemento a elemento;
- Subtração entre matrizes, elemento a elemento;
- Multiplicação entre matrizes;

Teste cada nova operação.

4 Árvores - ABP

4.1 Escreva uma função que encontre o **maior** valor existente em uma ABP.

4.2 Escreva uma função que encontre o **menor** valor existente em uma ABP.

4.3 Considerando uma ABP formada por caracteres, desenhe a árvore resultante da inserção dos seguintes elementos (nesta ordem):

- A, B, C, L, H, X, R, U, F, M, I
- I, D, T, U, A, F, E, N, Z, C, B, L

Você pode usar o site (https://app.diagrams.net/) para desenhar sua árvore.

4.4 Dado o **TAD: Árvore Binária de Pesquisa (ABP)** visto em aula, modifique a operação de remoção recursiva, de forma que a estratégia de remoção para o caso 3 (quando o nó possui duas subárvores) passe para o caso 3.2 do slide, ou seja, deve-se utilizar o nó com o menor valor da subárvore direita. Mostre apenas o código da nova implementação.

4.5 Dada uma Árvore Binária de Pesquisa, definida conforme o TAD: ABP visto em sala, crie uma função de caminhamento, que imprima os elementos da ABP de forma **decrescente**.

4.6 Modifique o **TAD: Árvore Binária de Pesquisa (ABP)** visto em aula, para incluir um novo ponteiro no **struct NO**. Este novo ponteiro deverá apontar para o PAI imediato, ou seja, além dos campos, *info*, *esq* e *dir*, agora cada nó deverá conter a referência do seu ***pai***.

Faça as modificações que se fizerem necessárias em todas as operações existente e inclua uma nova operação da seguinte forma: Imprima todos os caminhos (os valores) de um nó folha até a raiz, ou seja, em uma visitação, sempre que atingir um nó folha, a operação será acionada;

Teste novamente seu novo TAD usando a ideia do menu (do..while) com as seguintes opções, chamando as operações:

- 1 - Criar ABP;
- 2 - Inserir um elemento;
- 3 - Buscar um elemento
- 4 - Remover um elemento;
- 5 - Imprimir a ABP em ordem;
- 6 - Imprimir a ABP em pré-ordem;
- 7 - Imprimir a ABP em pós-ordem;
- 8 - Mostrar a quantidade de nós na ABP;
- 9 - Mostra todos os caminhos de nós folha até a raiz;
- 10 - Destruir a ABP;
- 11 - Sair;

4.7 Vá no *ChatGPT*, digite o seguinte:

> Dada uma ABP denotada por:
>
> ```
> typedef struct NO{
>     int info;
>     struct NO* esq;
>     struct NO* dir;
> }NO;
> typedef struct NO* ABP;
> ```
>
> Me mostre as 3 funções de caminhamento, em-ordem, pré-ordem e pós-ordem usando iteratividade em Linguagem C.

Teste os códigos gerados e compare com as versões recursivas do TAD. Deu para notar uma ENORME diferença entre usar Recursão e Iteração para os caminhamentos? Justifique. Inclua os códigos que foram resposta do *ChatGPT*.

5 Árvores AVL

5.1 Considerando seguinte ordem de inserção em uma Árvore AVL, diga quais serão as rotações que deverão acontecer para manter a árvore final balanceada, faça o desenho da árvore balanceada corrente a cada nova inserção e o desenho da árvore balanceada final:

```
50, 40, 45, 70, 80, 30, 20, 75, 78, 35, 31, 49, 42
```

5.2 Considerando o TAD: Árvore AVL visto em sala de aula, vamos resolver um problema da plataforma **BeeCrowd** da seguinte forma:

- Entre na página da nossa disciplina no menu Academic do beecrowd e acesse o problema na Lista de exercícios 2 (AVL) - Prova2, será o problema de número 1260;
- Interprete o problema e inclua as próximas modificações no nosso TAD para resolvê-lo:
- **Primeira Modificação:** Essa modificação, já foi feita na aula prática, o campo info do NO, passará a receber strings ao invés de inteiros, sendo assim altere as operações de inserção, remoção, busca e impressão.
- **Segunda Modificação:** A árvore AVL, assim como a ABP, não aceita elementos repetidos, o que será necessário para resolver o problema em questão. Sendo assim, adicione na struct NO, um campo qtd, para representar a quantidade de um elemento, dessa forma, ao ser inserido pela primeira vez, o elemento terá sua quantidade 1, caso ele seja inserido novamente, a operação agora só faz `qtd++` naquele nó. Use a informação da quantidade e altere as operações de inserção e remoção conforme a nova ideia.
- **Terceira Modificação:** Uma modificação deverá ser realizada na operação de impressão **em_ordem**. Para se adequar a saída esperada pelo problema. Já sabemos que a função **em_ordem** imprime os elementos de forma crescente, mas qual modificação será necessária segundo a nova chamada da função Imprime no trecho de código abaixo, e para se adequar a saída esperada pelo problema?
- **Ultima Modificação:** No caso, pela plataforma, todo o código deve estar presente em apenas 1 arquivo, sendo assim, todo o trecho de código relativo ao TAD deve estar presente no arquivo Main.c, antes da `int main()`. Faça essa mudança e use a função `int main()` a seguir para fazer a leitura das informações conforme a entrada definida pelo problema 1260:

Função Main, explicação abaixo:

```
int main(){
    int n, i;
    scanf("%d", &n);
    while(getchar()!= '\n');//Remove o \n da leitura do inteiro.
    char str[100];
    int caso = -1;
    int primeiro = 1;
    for(i=0; i<n; i++){
        AVL* A = criaAVL();
        int total = 0;
        //printf("Caso %d:\n", i+1);
        while(fgets(str, 100, stdin) != NULL){
            if(str[0] == '\n'){
                caso++;
                if(caso > i) break;
            }else{
                int tam = strlen(str);
                if(str[tam-1] == '\n') str[tam-1] = '\0';
                //tam = strlen(str);
                //printf("%s : %d\n", str, tam);
                insereElem(A, str);
                total++;
            }
        }

        if(primeiro){
            primeiro = 0;
        }else printf("\n");
        imprime(A, total);
        destroiAVL(A);
    }
    return 0;
}
```

(1) A entrada mostra um número de caso de testes, e para cada caso, um número indefinido de strings separados por uma linha em branco. Sendo assim a função main se comporta para ler cada caso de teste da seguinte maneira.
(2) A cada novo caso de teste, uma árvore AVL é criada.
(3) Sabe-se que o fgets lê toda uma string inclusive o $\backslash n$.
(4) Assim, quando ele ler uma linha em branco, o teste ($str[0] ==' \backslash n'$) será verdade, então representa o início de um novo caso de teste e cada string será lida.
(5) Como a leitura inclui o $\backslash n$ na string, um teste é feito para subtituí-lo pelo caracter $\backslash 0$ e o novo elemento é inserido na AVL.
(6) A repetição while para quando, começa inicia-se um novo caso de teste, ou atinge o fim de arquivo.
(7) Por fim, cada árvore final (cada caso de teste) é imprimida conforme a saída esperada e com um salto de linha entre as impressões.