Nome: **Prontuário:**

Data: / /

Experimento 1 - Pós-Teste - Estrutura de Dados

Instruções do Experimento

- Identifique de qual grupo você faz parte (**GC, GE1 ou GE2**).
- O sorteio dos grupos foi realizado em sala de aula.
- **Questões teóricas**:
  - GC → não poderá consultar nenhuma fonte.
  - GE1 → poderá utilizar apenas o **ChatGPT**.
  - GE2 → poderá utilizar apenas o **Gemini**.
- **Questão prática**:
  - GC → poderá consultar apenas **material próprio**.
  - GE1 → deverá utilizar exclusivamente o **ChatGPT**.
  - GE2 → deverá utilizar exclusivamente o **Gemini**.
- Esta avaliação **não impactará diretamente nas notas finais** da disciplina; ela conta apenas como **nota de participação**.
- O descumprimento das regras implicará em **perda da nota de participação**.
- Ao final, responda ao **Questionário Complementar**.

Recursividade

1. O que acontece se uma função recursiva não tiver um caso base?

   Entra em loop infinito / Estouro de pilha (stack overflow)

Arrays

2. Considere o seguinte código em C++:

```
int arr[6] = {7, 3, 4, 2, 9, 6}
arr[4] = arr[4] + 1;
for (int i = 0; i <= 4; i+2) {
    cout << arr[i] << "␣";
}
```

   Qual a saída desse código?

   7 4 10

Listas

3. Em uma lista encadeada circular, o ponteiro *"próximo"* do último nó aponta para onde?

   O último nó aponta para o primeiro nó.

Pilha e Fila

4. Em uma estrutura de dados do tipo fila (queue), a ordem de acesso aos elementos é:

   a) FIFO
   b) LIFO
   c) FILO
   d) LILO

Árvores

5. Considerando os conceitos de Árvores. Defina os termos:

A. Raiz
O nó principal da árvore; o topo da hierarquia.

B. Pais
Nós diretamente conectados a um ou mais nós inferiores.

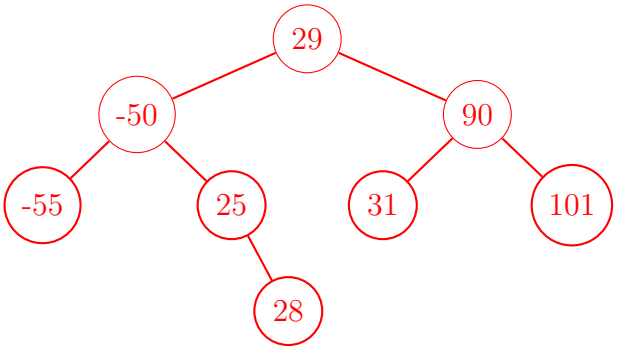
C. Nível de um nó
Distância da raiz até esse nó.

D. Subárvore
Qualquer porção da árvore formada por um nó e seus descendentes.

Árvore Binária de Busca

6. Considere a seguinte sequência de inserções em uma árvore binária de busca vazia: 30, -35, 90, 101, -50, 31, 25, -55, 29, 28, remova os nós 30 e -35 caso necessário use o conceito de substituir o nó a ser retirado pelo nó mais à direita da subárvore da esquerda.

A. Desenhe a árvore resultante após todas as inserções e remoções.



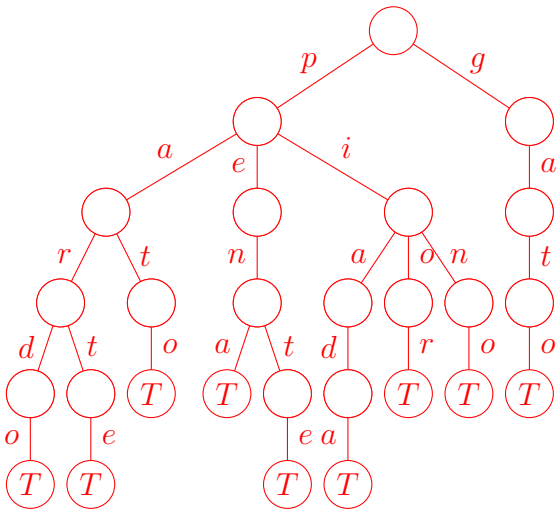
B. Qual é o percurso pós-ordem da árvore final?
-55, 28, 25, -50, 31, 101, 90, 29

C. Qual é o percurso pré-ordem da árvore final?
29, -50, -55, 25, 28, 90, 31, 101

D. Qual é o percurso em largura (BFS) da árvore final? 29, -50, 90, -55, 25, 31, 101, 28

Tries

7. Dado o conjunto de palavras { ”pato”, ”pardo”, ”parte”, ”pente”, ”pena”, ”pior”, ”piada”, ”pino”, ”gato”}, desenhe uma Trie representando todas elas.



Árvore AVL

8. Preencha com (V) Verdadeiro ou (F) Falso:

a) (F) O fator de balanceamento de um nó numa árvore AVL que vai ser balanceada pode chegar a 3.

b) (F) Árvores AVL é um tipo de Árvore N-ária.

c) (V) Inserções e remoções podem causar desbalanceamento, exigindo rotações.

d) (F) Uma árvore AVL tenta evitar degeneração em forma de lista, mas pode acontecer.

Prática

9. Exercício 1 – Árvore AVL de Letras.

Implemente um programa que trabalhe com árvores AVL da seguinte forma:

- A árvore deve ser inicializada contendo, em ordem de inserção, as letras do **seu primeiro nome**.
- O programa deve receber como entrada uma **palavra digitada pelo usuário**.
- As letras dessa palavra devem ser inseridas na árvore, considerando apenas as **letras não repetidas**.
- Após todas as inserções, a árvore resultante deve ser exibida em **percurso em largura** (nível a nível).

Exemplo de função para ler cada caractere de uma *string*:

```
string texto = "Exemplo";
for (char c : texto) {
    // "c" armazena cada caractere
    da string texto.
}
```

```
// Primeiro peguei o código base de árvore AVL, disponível no classroom,
// Depois adaptei para fazer o que o exercício pede.
#include <iostream>
using namespace std;

struct NoAVL
{
    // Altere o tipo da chave para char
    char chave;
    int fb;
    NoAVL* dir;
    NoAVL* esq;
};

// Peguei o código base de uma ABB, disponível no classroom,
// para usar a função de percurso em largura.
NoAVL* fila[100]; // Aumentei para evitar problemas com muitas letras.
int inicio = 0;
int fim = 0;

void Enfileirar(NoAVL* valor){
    if (fim == 100)
    {
        cout << "Fila cheia!!" << endl;
        return;
    }

    fila[fim] = valor;
    fim++;
}

void Desenfileirar(){
    if (inicio == fim){
        cout << "Fila vazia!!" << endl;
        return;
    }

    inicio++;
}

// Ajustei a função Largura para percorrer a lista maior
void Largura(NoAVL* raiz){
    if (!raiz) return;
```

```
    inicio = 0;
    fim = 0;
    Enfileirar(raiz);

    while (inicio < fim){
        NoAVL* atual = fila[inicio];
        inicio++;

        cout << atual->chave << " - ";

        if (atual->esq) Enfileirar(atual->esq);
        if (atual->dir) Enfileirar(atual->dir);
    }
}

// Ajustei a função CriarNo para o tipo char
NoAVL* CriarNo(char chave){
    NoAVL* novoNo = new NoAVL;
    novoNo->chave = chave;
    novoNo->dir = nullptr;
    novoNo->esq = nullptr;
    novoNo->fb = 0;
    return novoNo;
}

NoAVL* RotacaoL(NoAVL* p){
    NoAVL* u = p->esq;

    if (u->fb == -1)
    {
        /* Rotação LR */
        NoAVL* v = u->dir;
        u->dir = v->esq;
        v->esq = u;
        p->esq = v->dir;
        v->dir = p;

        if (v->fb == 1) {
            u->fb = -1;
            p->fb = 0;
        } else if (v->fb == -1) {
            u->fb = 0;
            p->fb = 1;
        } else {
            u->fb = 0;
            p->fb = 0;
        }

        v->fb = 0;
```

```
        return v;
    }

    /* Rotação LL */
    p->esq = u->dir;
    u->dir = p;
    p->fb = 0;
    u->fb = 0;

    return u;
}

NoAVL* RotacaoR(NoAVL* p){
    NoAVL* u = p->dir;

    if (u->fb == 1)
    {
        /* Rotação RL */
        NoAVL* v = u->esq;
        u->esq = v->dir;
        v->dir = u;
        p->dir = v->esq;
        v->esq = p;

        if (v->fb == 1) {
            p->fb = -1;
            u->fb = 0;
        } else if (v->fb == -1) {
            p->fb = 0;
            u->fb = 1;
        } else {
            p->fb = 0;
            u->fb = 0;
        }

        v->fb = 0;

        return v;
    }

    /* Rotação RR */
    p->dir = u->esq;
    u->esq = p;
    p->fb = 0;
    u->fb = 0;

    return u;
}

// Ajustei a função Inserir para o tipo char
```

```
NoAVL* Inserir (NoAVL*& raiz, char chave, bool& cresceu){

    if(raiz == nullptr){
        raiz = CriarNo(chave);
        cresceu = true;
        return raiz;
    }

    else if(chave < raiz->chave){
        Inserir (raiz->esq, chave, cresceu);
        if (cresceu){
            if (raiz->fb == 0)
                raiz->fb = 1;
            else if (raiz->fb == -1)
                raiz->fb = 0, cresceu = false;
            else if (raiz->fb == 1) {
                raiz = RotacaoL (raiz);
                cresceu = false;
            }
        }
    }
    else {
        Inserir (raiz->dir, chave, cresceu);
        if (cresceu){
            if (raiz->fb == 0)
                raiz->fb = -1;
            else if (raiz->fb == 1)
                raiz->fb = 0, cresceu = false;
            else if (raiz->fb == -1){
                raiz = RotacaoR (raiz);
                cresceu = false;
            }
        }
    }

    return raiz;
}

// Alterei a função Buscar para rerotnar um bool e usar char
bool Buscar (char chave, NoAVL* raiz){
    if (raiz == nullptr){
        return false;
    }
    else if (raiz->chave == chave){
        return true;
    }
    else if(chave < raiz->chave)
        return Buscar (chave, raiz->esq);
    else
        return Buscar (chave, raiz->dir);
```

```
}

int main()
{
    NoAVL* raiz = nullptr;
    bool cresceu = false;

    // Inicializa a árvore com as letras do nome "IGOR"
    Inserir(raiz, 'I', cresceu);
    Inserir(raiz, 'G', cresceu);
    Inserir(raiz, 'O', cresceu);
    Inserir(raiz, 'R', cresceu);

    // Recebe uma palavra do usuário
    string palavra;

    cout << "Digite uma palavra: ";
    cin >> palavra;

    // Insere as letras da palavra na árvore, ignorando letras repetidas
    for (char letra : palavra) {
        // Verifica se a letra já está na árvore
        if (!Buscar(toupper(letra), raiz))
        {
            // (opcional) toupper para que todas as letras sejam maiúsculas
            Inserir(raiz, toupper(letra), cresceu);
        }
    }

    // Exibe a árvore em percurso em largura
    cout << "Árvore em percurso em largura: ";
    Largura(raiz);
    cout << endl;

    return 0;
}
```