CEFET - RJ

Centro Federal de Educação Tecnológica Celso Suckow da Fonseca

AEDS 2 -Relatório *Final*

Sistema de atendimento com Árvore-Rn: Testes e Análise de complexidade

---

Carlos Eduardo Dias Silva
Anisio Matheus Alves Da Fonseca

Laura Assis (docente)

5 de maio de 2025

Sumário

1 Introdução

Este relatório apresenta o desenvolvimento de uma aplicação em linguagem C que implementa uma árvore rubro-negra funcional. O sistema de atendimento trata as entradas e, a partir de uma lógica definida, determina qual procedimento será executado na árvore.

Foram implementadas as funções de inserção, remoção e exibição da árvore.

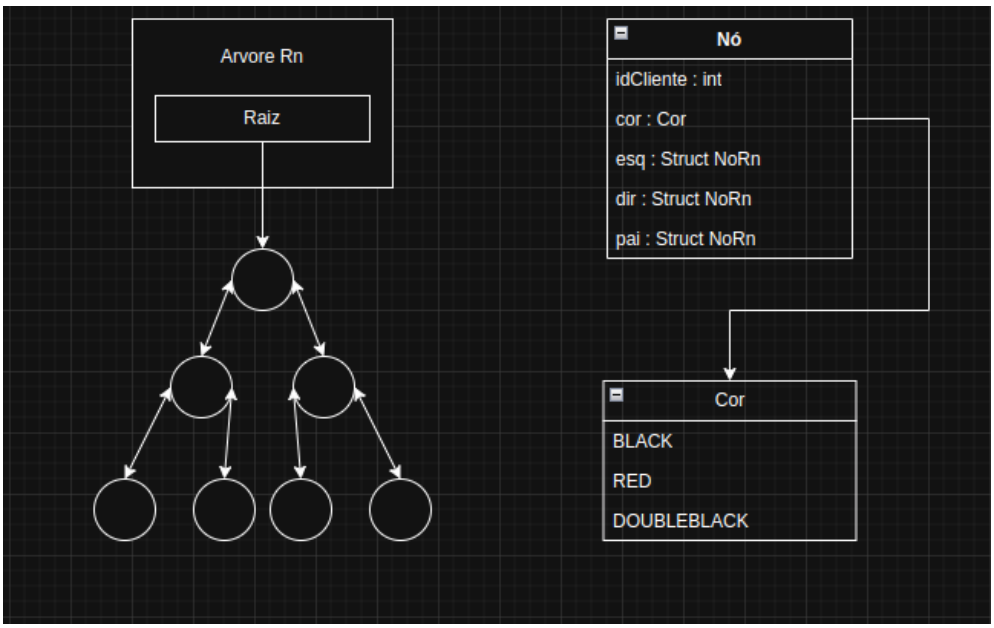


Figura 1: Diagrama da estrutura

A árvore rubro-negra armazena uma **raiz** (ponteiro para um nó), que contém:

- `idCliente`;
- `cor` (struct dedicada);
- ponteiros para o **pai**, **filho esquerdo** e **filho direito**.

Essas estruturas serviram de base para o desenvolvimento do sistema de atendimento.

2 Metodologia

Utilizando a linguagem C, foram desenvolvidas funções para atender aos requisitos funcionais do trabalho. Para controle de versionamento e trabalho colaborativo, utilizou-se a plataforma *GitHub*.

2.1 Inserção

A função de inserção utiliza uma **função auxiliar** para aplicar os três casos de balanceamento de **árvores rubro-negras**, garantindo que as propriedades da estrutura sejam mantidas após a operação.

```
/*
@brief aplica os 3 casos de balanceamento
@param novoNo: no usado na insercao
@param arvore: arvore rubro negra

*/
bool ajustarInsercao(NoRn *novoNo, ArvoreRn *arvore);

/*
@brief insere um novo NoRn na arvore      You, 1 second ag
@param idCLiente: numero de registro do cliente
@param arvore: arvore rubro negra

*/
bool inserirNo(int idCliente, ArvoreRn *arvore);
```

Figura 2: Cabeçalho da função, presente em ArvoreRn.h

2.1.1 Balanceamento da inserção

Realiza o balanceamento, aplicando os casos para um nó com nenhum, um ou dois filhos.

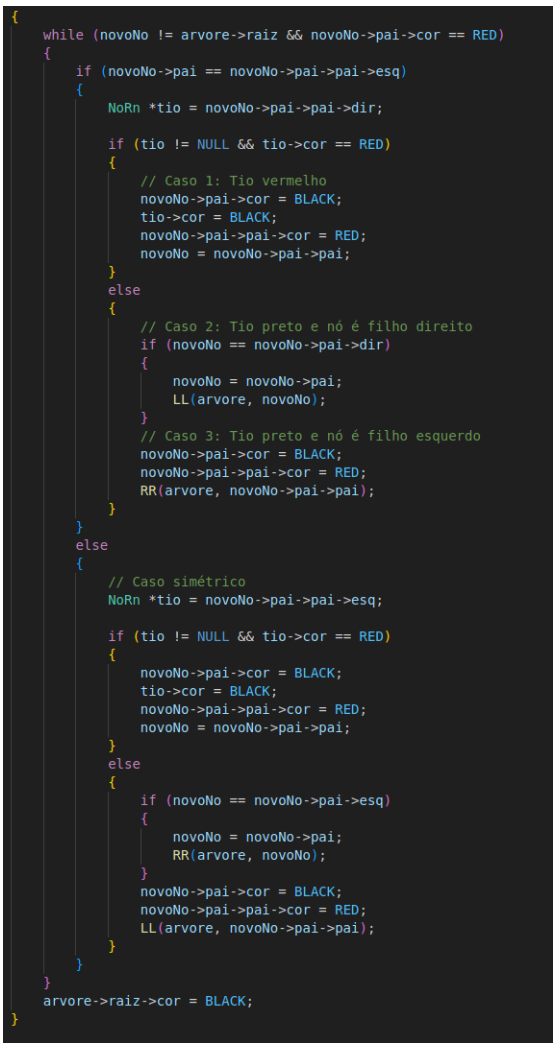
2.2 Remoção

A função de remoção também utiliza uma **função auxiliar** para aplicar o balanceamento da **árvore rubro-negra**, assegurando que suas propriedades sejam preservadas após a operação.
Além disso, faz uso das seguintes operações:

- **Rotações** desacopladas: RR, LL, RL e LR;
- Função de **nó antecessor** (para encontrar o predecessor in-order);
- Operação de **transplante entre subárvores** (para reorganização da árvore).

2.2.1 Balanceamento da remoção

Realiza o balanceamento aplicando os casos para um nó com dois filhos. Para nós com nenhum ou apenas um filho (casos 1 e 2), o tratamento é feito diretamente na função **RemoverNoRn**.



```
{
    while (novoNo != arvore->raiz && novoNo->pai->cor == RED)
    {
        if (novoNo->pai == novoNo->pai->pai->esq)
        {
            NoRn *tio = novoNo->pai->pai->dir;

            if (tio != NULL && tio->cor == RED)
            {
                // Caso 1: Tio vermelho
                novoNo->pai->cor = BLACK;
                tio->cor = BLACK;
                novoNo->pai->pai->cor = RED;
                novoNo = novoNo->pai->pai;
            }
            else
            {
                // Caso 2: Tio preto e nó é filho direito
                if (novoNo == novoNo->pai->dir)
                {
                    novoNo = novoNo->pai;
                    LL(arvore, novoNo);
                }
                // Caso 3: Tio preto e nó é filho esquerdo
                novoNo->pai->cor = BLACK;
                novoNo->pai->pai->cor = RED;
                RR(arvore, novoNo->pai->pai);
            }
        }
        else
        {
            // Caso simétrico
            NoRn *tio = novoNo->pai->pai->esq;

            if (tio != NULL && tio->cor == RED)
            {
                novoNo->pai->cor = BLACK;
                tio->cor = BLACK;
                novoNo->pai->pai->cor = RED;
                novoNo = novoNo->pai->pai;
            }
            else
            {
                if (novoNo == novoNo->pai->esq)
                {
                    novoNo = novoNo->pai;
                    RR(arvore, novoNo);
                }
                novoNo->pai->cor = BLACK;
                novoNo->pai->pai->cor = RED;
                LL(arvore, novoNo->pai->pai);
            }
        }
    }
    arvore->raiz->cor = BLACK;
}
```

Figura 3: Lógica para o balanceamento

2.3 Gerenciamento do Atendimento

Duas funções foram implementadas: uma para preencher a fila de atendimento com os dados de entrada e outra para processar esses dados e realizar as operações necessárias. A primeira função adiciona novos clientes à fila, baseando-se no ID fornecido. Quando um cliente é inserido, ele é colocado no final da fila. Caso a entrada seja -1, a segunda função é acionada, removendo o cliente mais antigo da fila, o que é feito utilizando a árvore rubro-negra para buscar e remover o nó correspondente ao cliente mais antigo. A árvore rubro-negra garante que a remoção do cliente mais antigo seja eficiente, mantendo o balanceamento da estrutura durante o processo.

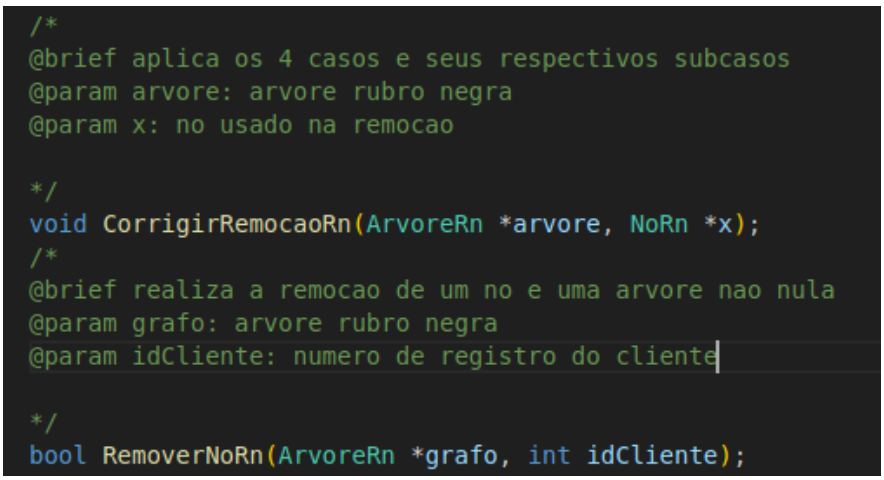
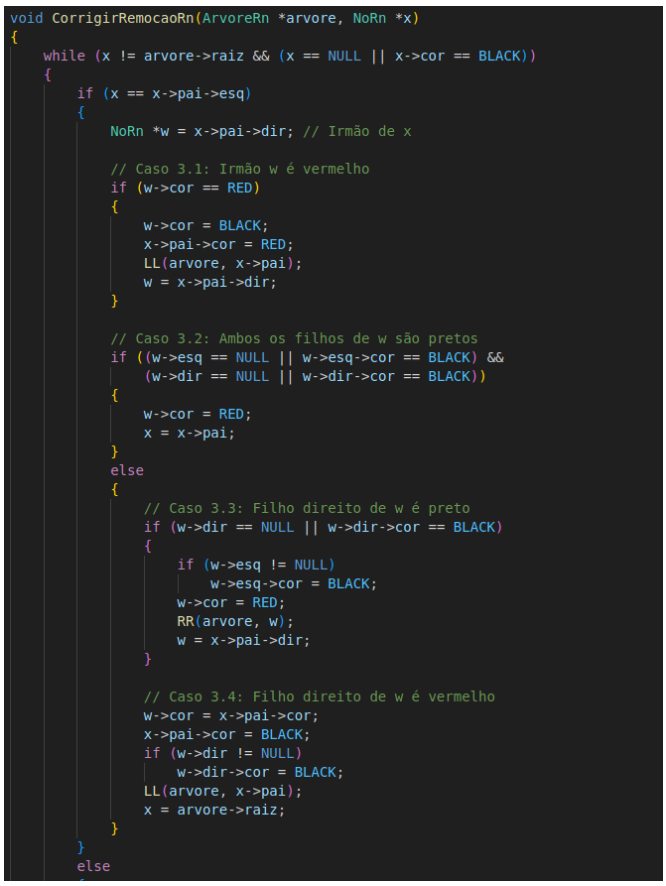


Figura 4: Cabeçalho da função, presente em ArvoreRn.h

2.4 Testes e Análise Temporal

Para os testes, foi utilizado o sistema Linux, na distribuição Ubuntu 22.04 LTS. O código foi segmentado em dois arquivos: um arquivo `.h`, contendo as `structs`, bibliotecas, cabeçalhos das funções e comentários relevantes para o entendimento; e outro arquivo com a implementação principal.

Foi desenvolvida uma análise gráfica para demonstrar o desempenho do algoritmo em relação ao volume de dados, utilizando uma escala logarítmica.



```c
void CorrigirRemocaoRn(ArvoreRn *arvore, NoRn *x)
{
    while (x != arvore->raiz && (x == NULL || x->cor == BLACK))
    {
        if (x == x->pai->esq)
        {
            NoRn *w = x->pai->dir; // Irmão de x

            // Caso 3.1: Irmão w é vermelho
            if (w->cor == RED)
            {
                w->cor = BLACK;
                x->pai->cor = RED;
                LL(arvore, x->pai);
                w = x->pai->dir;
            }

            // Caso 3.2: Ambos os filhos de w são pretos
            if ((w->esq == NULL || w->esq->cor == BLACK) &&
                (w->dir == NULL || w->dir->cor == BLACK))
            {
                w->cor = RED;
                x = x->pai;
            }
            else
            {
                // Caso 3.3: Filho direito de w é preto
                if (w->dir == NULL || w->dir->cor == BLACK)
                {
                    if (w->esq != NULL)
                        w->esq->cor = BLACK;
                    w->cor = RED;
                    RR(arvore, w);
                    w = x->pai->dir;
                }

                // Caso 3.4: Filho direito de w é vermelho
                w->cor = x->pai->cor;
                x->pai->cor = BLACK;
                if (w->dir != NULL)
                    w->dir->cor = BLACK;
                LL(arvore, x->pai);
                x = arvore->raiz;
            }
        }
        else
```

Figura 5: Lógica para o balanceamento

```c
ArvoreRn *tratarInputDados(FilaAtendimento *fila)
{
    ArvoreRn *grafo = criaArvore();

    for (int i = 0; i < fila->totalAtendimentos; i++)
    {

        if (fila->vetor[i] != -1)
        {
            inserirNo(fila->vetor[i], grafo);
            printf("ins: %d\n", fila->vetor[i]);
        }
        else
        {
            NoRn *aux = menorNo(grafo->raiz);
            printf("rem: %d\n", aux->idCliente);
            RemoverNoRn(grafo, aux->idCliente);
        }       You, 4 days ago • funcoes auxiliares e main …
    }
    return grafo;
}

void insercaoNaFila(FilaAtendimento *fila)
{

    fila->vetor = malloc(sizeof(int) * MAX);
    int aux = 0;
    while (scanf("%d", &aux) == 1)
    {
        fila->vetor[fila->totalAtendimentos] = aux;
        fila->totalAtendimentos++;
    }
}
```

Figura 6: Lógica para o atendimento

```
cadudias@cadudias-IdeaPad-Gaming-3-15IMH05:~/Área de trabalho/AEDS2/TrabalhoRn/Arvore-Rn$ gcc ArvoreRnImp.c -o saida
cadudias@cadudias-IdeaPad-Gaming-3-15IMH05:~/Área de trabalho/AEDS2/TrabalhoRn/Arvore-Rn$ ./saida < 3.in
100 BLACK
30 BLACK
29 RED
50 BLACK
40 RED
70 BLACK
63 RED
80 RED
150 RED
120 BLACK
130 RED
190 BLACK
cadudias@cadudias-IdeaPad-Gaming-3-15IMH05:~/Área de trabalho/AEDS2/TrabalhoRn/Arvore-Rn$
```

Figura 7: Compilação e execução da implementação

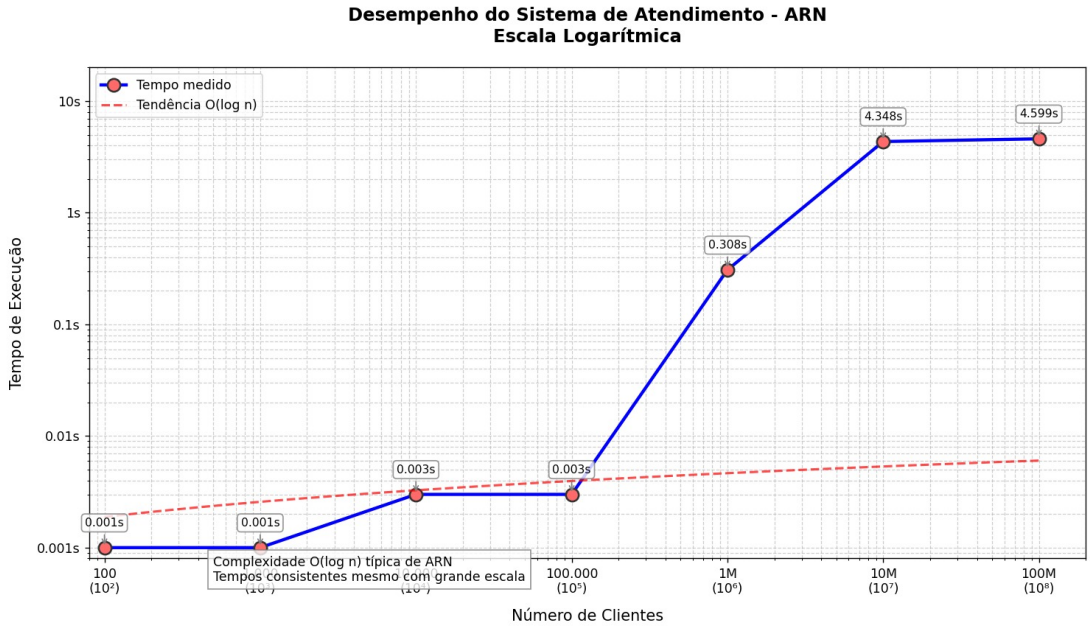


Figura 8: Gráfico

3 Conclusão

A implementação de uma árvore rubro-negra em linguagem C demonstrou-se eficaz para o gerenciamento dinâmico de dados em um sistema de atendimento baseado em fila. As operações de inserção e remoção foram cuidadosamente estruturadas para manter as propriedades essenciais da árvore, com apoio de funções auxiliares e lógica de balanceamento rigorosa.

Além disso, a divisão modular do código, o uso de boas práticas de programação e o suporte a testes no ambiente Linux reforçaram a robustez e a clareza do desenvolvimento. Dessa forma, o sistema atendeu aos requisitos propostos, consolidando-se como uma solução eficiente para manipulação de dados com balanceamento automático.

4 Referências

[1] Carlos Dias. Arvore-Rn. Disponível em: https://github.com/CarlosDiasS/Arvore-Rn