
Universidade Federal de Roraima

# INSERÇÃO DA ÁRVORE RED AND BLACK

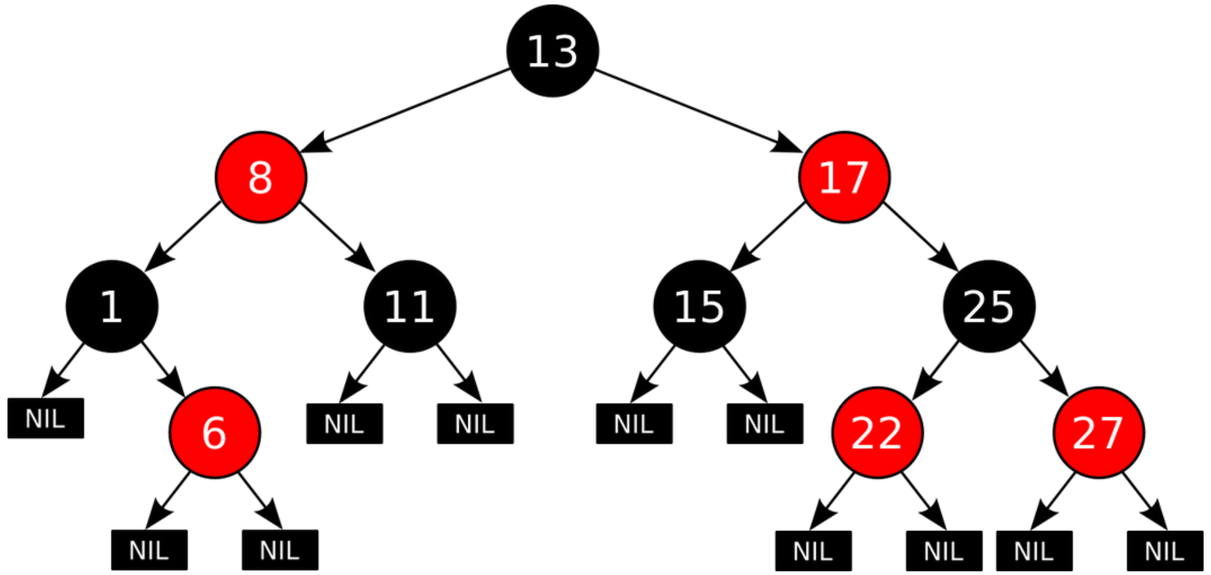
**Filipe Gabriel,**
**Marcus Vinícius**

# Introdução

- **Estruturas de dados são fundamentais para organização e manipulação eficiente de grandes volumes de dados.**

- **A Árvore Red-Black é uma árvore binária de busca balanceada que garante operações eficientes de busca, inserção e remoção.**

**Objetivo: analisar a inserção na Árvore Red-Black em termos de funcionalidade, custo computacional e comparar com outra estrutura eficiente: a tabela Hash .**



# Estrutura da Árvore Red-Black

**Características principais:**

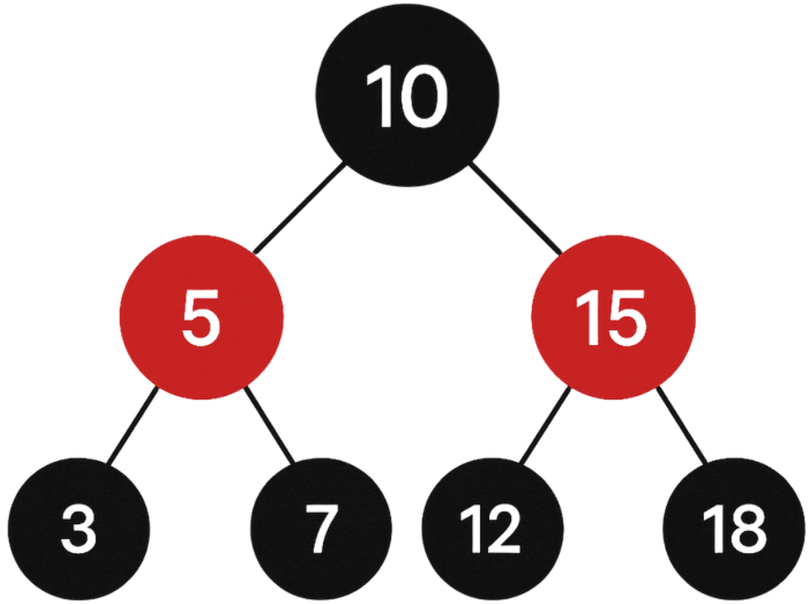
- **É uma árvore binária de busca com balanceamento garantido.**
- **Cada nó é vermelho ou preto.**

**Propriedades fundamentais:**

- **Raiz sempre preta.**
- **Nenhum nó vermelho pode ter um filho vermelho.**
- **Todos os caminhos de um nó até suas folhas possuem o mesmo número de nós pretos (propriedade de altura preta).**

**Benefícios:**

**Busca, inserção e remoção eficientes com complexidade  O(logn).**

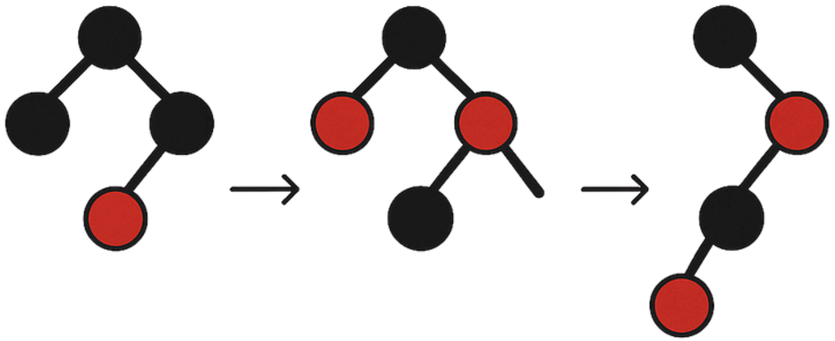



UFRR

# Inserção na Árvore Red-Black

**Explicação do processo:**

- **Inserção inicial: Insere o novo nó seguindo as regras de uma árvore binária de busca.**
- **Correção de violações: Corrige violações de propriedades através de recolorações e rotações.**



**Operações envolvidas:**

- **Rotação à esquerda/direita.**
- **Recoloração de nós.**

# Implementação da Inserção

## Função de inserção (insert)

```c
void insert(Node **root, int key)
{
    Node *z = createNode(key);
    Node *y = NULL;
    Node *x = *root;

    while (x)
    {
        y = x;
        x = (z->key < x->key) ? x->left : x->right;
    }
    z->parent = y;
    if (!y)
        *root = z;
    else if (z->key < y->key)
        y->left = z;
    else
        y->right = z;

    fixViolation(root, z);
}
```

## Correção de Violações (fixViolation)

```c
fixViolation(Node **root, Node *z)

while (z != *root && z->parent->color == RED)
{
    Node *gp = z->parent->parent;
    if (z->parent == gp->left)
    {
        Node *y = gp->right;
        if (y && y->color == RED)
        {
            z->parent->color = BLACK;
            y->color = BLACK;
            gp->color = RED;
            z = gp;
        }
        else
        {
            if (z == z->parent->right)
            {
                z = z->parent;
                rotateLeft(root, z);
            }
            z->parent->color = BLACK;
            gp->color = RED;
            rotateRight(root, gp);
        }
    }
    else
    {
        Node *y = gp->left;
        if (y && y->color == RED)
        {
            z->parent->color = BLACK;
            y->color = BLACK;
            gp->color = RED;
            z = gp;
        }
        else…
    }
}
(*root)->color = BLACK;
```

UFRR

# Implementação da Inserção

## Exemplo de Rotação

```
void rotateLeft(Node **root, Node *x)
{
    Node *y = x->right;
    x->right = y->left;
    if (y->left)
        y->left->parent = x;
    y->parent = x->parent;
    if (!x->parent)
        *root = y;
    else if (x == x->parent->left)
        x->parent->left = y;
    else
        x->parent->right = y;
    y->left = x;
    x->parent = y;
}
```

# Complexidade da Inserção

**Complexidade Teórica:**

- **A altura da Árvore Red-Black é no máximo 2log(n+1).**
- **Complexidade da inserção é O(logn).**

**Justificativa:**

- **A manutenção do balanceamento após a inserção requer um número limitado de operações de rotação e recoloração.**

UFRR

# Experimentos e Resultados

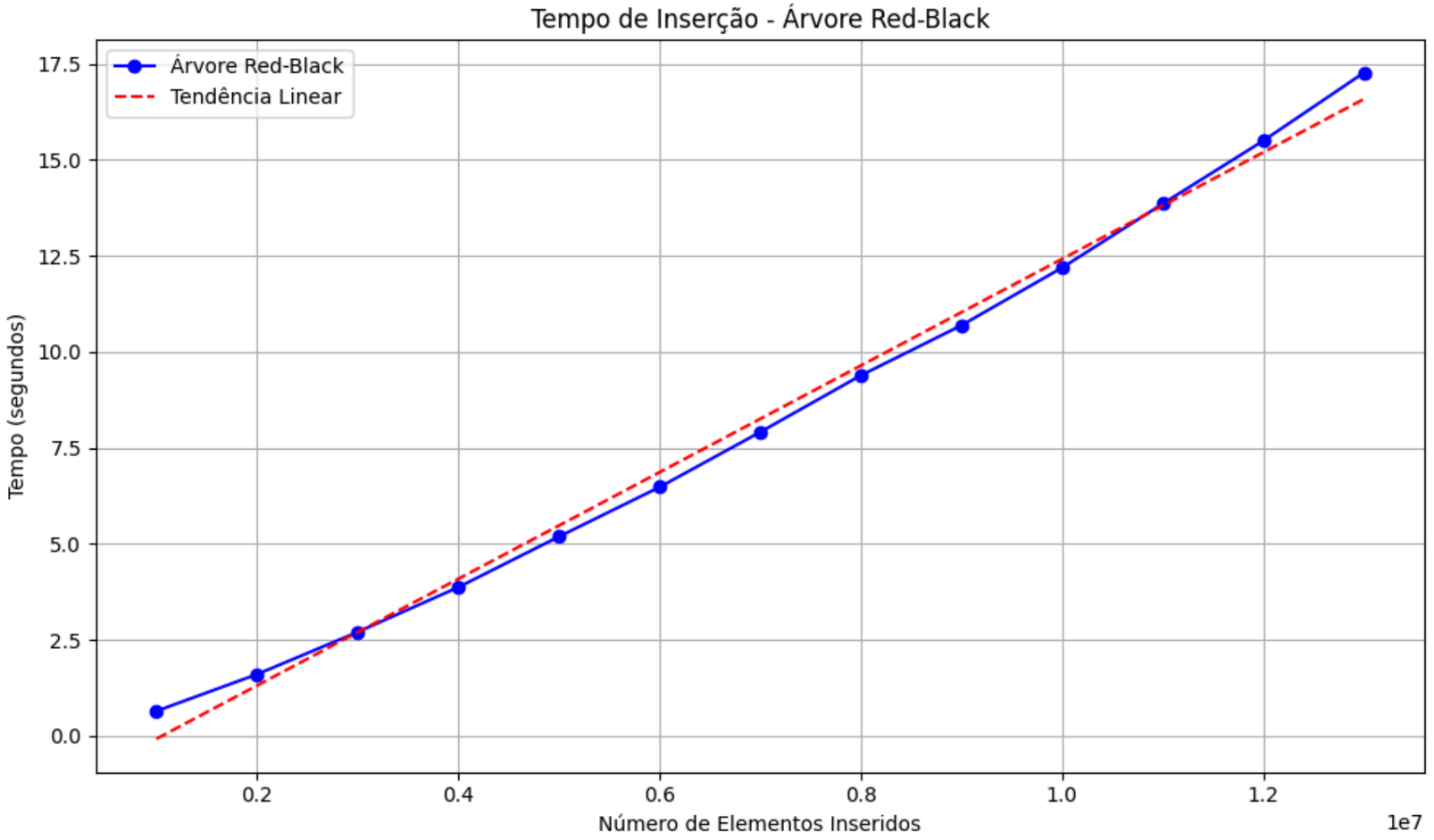
**Inserção de 1 até 13 milhões de elementos**

```
[mv@archlinux analise_algoritmos]$ ./algoritmo
Inserção de 1000000 elementos: 0.64389 segundos
Inserção de 2000000 elementos: 1.60182 segundos
Inserção de 3000000 elementos: 2.70279 segundos
Inserção de 4000000 elementos: 3.86975 segundos
Inserção de 5000000 elementos: 5.18800 segundos
Inserção de 6000000 elementos: 6.47804 segundos
Inserção de 7000000 elementos: 7.91111 segundos
Inserção de 8000000 elementos: 9.38703 segundos
Inserção de 9000000 elementos: 10.69200 segundos
Inserção de 10000000 elementos: 12.19318 segundos
Inserção de 11000000 elementos: 13.85909 segundos
Inserção de 12000000 elementos: 15.50094 segundos
Inserção de 13000000 elementos: 17.27467 segundos
```

UFRR

# Experimentos e Resultados

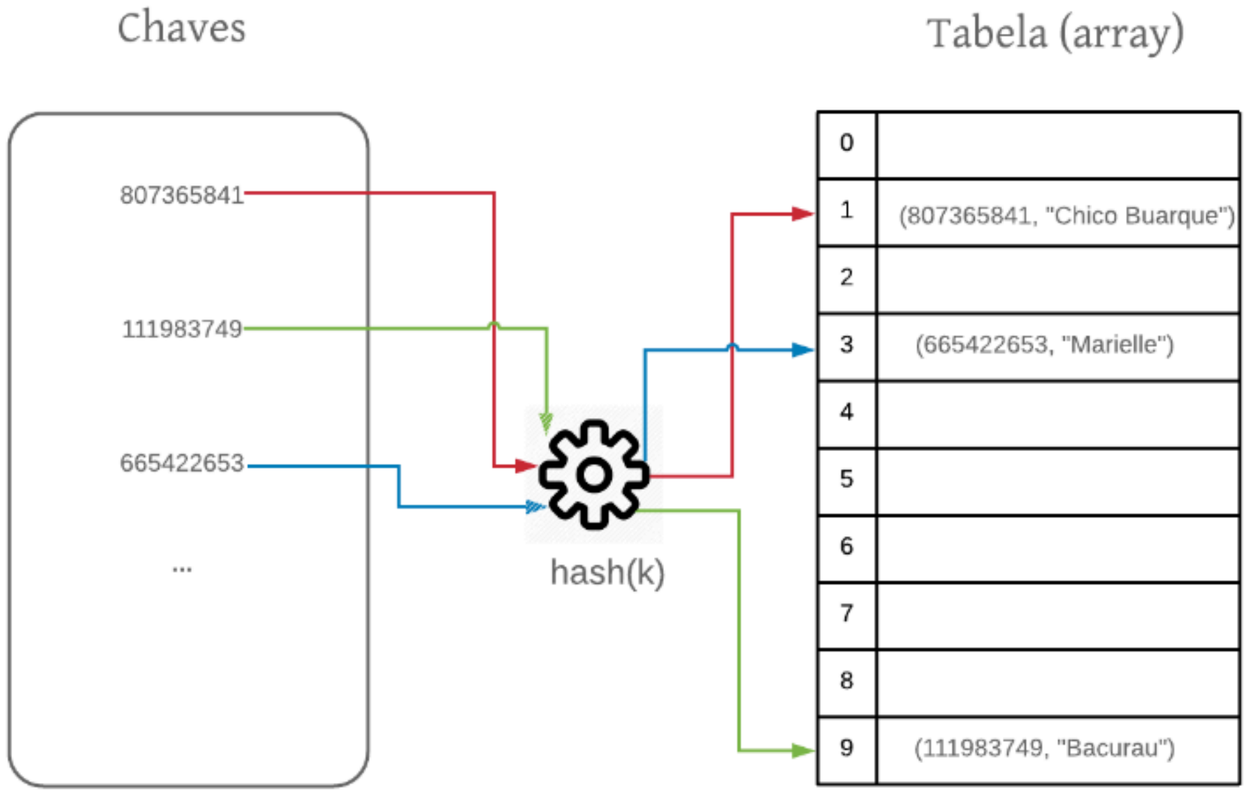
## Gŕafico do tempo de inserção



UFRR

# Comparação com Tabela Hash

- **Inserção mais rápida: O(1) médio vs O(log n).**
- **Menor custo de inserção em grandes volumes.**
- **Não mantém dados ordenados.**
- **Não permite buscas por menor ou maior elemento.**



UFRR

# Experimentos com Tabela Hash

**Inserção de 1 até 13 milhões de elementos**

```
[mv@archlinux algoritmos_eficientes]$ ./tabela_hash
Inserção de 1000000 elementos: 0.04244 segundos
Inserção de 2000000 elementos: 0.07570 segundos
Inserção de 3000000 elementos: 0.11153 segundos
Inserção de 4000000 elementos: 0.14611 segundos
Inserção de 5000000 elementos: 0.18118 segundos
Inserção de 6000000 elementos: 0.21820 segundos
Inserção de 7000000 elementos: 0.25720 segundos
Inserção de 8000000 elementos: 0.29483 segundos
Inserção de 9000000 elementos: 0.32612 segundos
Inserção de 10000000 elementos: 0.36554 segundos
Inserção de 11000000 elementos: 0.39911 segundos
Inserção de 12000000 elementos: 0.43212 segundos
Inserção de 13000000 elementos: 0.47136 segundos
```

UFRR

# Conclusão

**Resumo:**

- **Analisamos o processo de inserção na Árvore Red-Black, destacando sua eficiência e balanceamento.**
- **Comparada à Tabela Hash, a Árvore Red-Black é mais flexível, mas menos eficiente em inserções puras.**

**Reflexão:**

**A escolha da estrutura depende das necessidades da aplicação: ordenação vs eficiência de acesso.**

UFRR

# OBRIGADO PELA ATENÇÃO!!!!!!