

## 23.1\_Trabalho\_ESD

### Tema: Árvores Splay

Disciplina: Estrutura de Dados

Professora: Claudia Ferlin

**Grupo 1:**

Mylena Oliveira

Gabriela Santos

Guilherme Ribeiro

Lucas Sanginetto

Rio de Janeiro, 29 de junho de 2023

## **SUMÁRIO**

<b>Introdução.....</b>	<b>p.3</b>
<b>A estrutura da árvore.....</b>	<b>p.3</b>
<b>Como é feita a Inclusão?.....</b>	<b>p.5</b>
<b>Como é feita a Exclusão?.....</b>	<b>p.6</b>
<b>Exemplo de Código para Inclusão.....</b>	<b>p.8</b>
<b>Exemplo de Código para Exclusão.....</b>	<b>p.9</b>
<b>Exemplo de Código para Busca.....</b>	<b>p.10</b>
<b>Código geral da Splay Tree.....</b>	<b>p.11</b>
<b>Bibliografia.....</b>	<b>p.16</b>

## Introdução

A árvore Splay foi criada em 1985, por Daniel Sleator e Robert Tarjan. Seu objetivo principal é fazer com que o dado mais recentemente acessado seja movido para a raiz da árvore, “splayado”. As operações principais em uma árvore splay são executadas em tempo  $O(\log n)$ , em que  $n$  é o número de entradas na árvore.

A operação de splay, então, envolve uma série de rotações para trazer o nó desejado para a raiz. Os três tipos de rotações mais comuns que podem ser aplicadas dependendo da estrutura atual da árvore e da posição do nó que está sendo “splayado” são: rotação zig, rotação zig-zig e rotação zig-zag.

Rotação zig: É uma rotação simples onde um nó é trocado com seu pai. Isso é feito quando o nó desejado está a uma profundidade de um na árvore (ele é filho da raiz).

Rotação zig-zig: Duas rotações zig consecutivas são aplicadas para trazer o nó para a raiz. Isso ocorre quando o pai do nó desejado é filho da raiz, e ambos têm a mesma orientação (ambos são filhos esquerdos ou ambos são filhos direitos).

Rotação zig-zag: Duas rotações zig são aplicadas em sequência para trazer o nó para a raiz. Isso ocorre quando o nó e seu pai têm orientações opostas (um é filho esquerdo e o outro é filho direito).

A cada splay, a árvore é reorganizada para garantir que o nó “splayado” seja colocado na raiz. Isso melhora o tempo de acesso subsequente a esse nó, pois a raiz é o ponto de partida mais eficiente para futuras operações.

A operação de *splay* pode ser aplicada mesmo quando o elemento não está presente na árvore. Nesse caso, o último nó visitado durante a busca é “splayado”, mantendo a propriedade de otimização de acesso.

## A estrutura da árvore

A estrutura de uma árvore splay é parecida com a de uma árvore binária de pesquisa tradicional, com a adição de uma operação especial chamada “splay”. A operação de Splay é usada para trazer um nó específico para a raiz da árvore, reorganizando a árvore de acordo com algumas regras.

Basicamente, uma árvore splay é uma árvore binária de pesquisa autoajustável em que os nós mais acessados são movidos para a raiz, facilitando o acesso subsequente. Através da operação de splay, a árvore é reorganizada para manter os elementos frequentemente acessados próximos à raiz, reduzindo o tempo de acesso.

Já que a prioridade é manter os nós frequentemente acessados próximos à raiz, as árvores splay não precisam manter um balanceamento estrito após as operações. Embora

isso não seja um problema na maioria das situações, em alguns casos, a estrutura pode se degradar e a eficiência das operações pode diminuir.

### **Vantagens e desvantagens da árvore splay:**

#### **Vantagens:**

**Eficiência média:** As árvores splay oferecem um desempenho médio eficiente para operações de busca, inserção e remoção, especialmente em padrões de acesso dinâmicos. Isso ocorre porque os nós frequentemente acessados tendem a ficar mais próximos da raiz, melhorando o tempo de acesso.

**Simplicidade:** Comparadas a outras estruturas de dados balanceadas, como árvores AVL ou árvores rubro-negras, as árvores splay são mais simples de implementar e requerem menos sobrecarga de armazenamento, pois não precisam manter informações adicionais para o balanceamento.

**Adaptabilidade:** As árvores splay são auto ajustáveis e se adaptam aos padrões de acesso dos dados. Elas organizam (e reorganizam) a estrutura durante as operações, trazendo nós frequentemente acessados para a raiz. Isso permite que a árvore se ajuste dinamicamente às mudanças nos padrões de acesso.

#### **Desvantagens:**

**Pior caso:** Embora a árvore splay tenha um bom desempenho médio, o pior caso de uma operação pode ter um tempo de execução realmente alto. Em algumas situações, a sequência de rotações necessárias para trazer um nó para a raiz pode ser bastante longa, resultando em um tempo de execução menos eficiente.

#### **Exemplo de pior caso:**

Em uma árvore splay em que as chaves estão inseridas em ordem crescente ou decrescente, suponha que é desejável realizar uma busca para um nó com uma chave específica que está no extremo oposto da árvore em relação à raiz. Se a árvore splay estiver desbalanceada e o nó desejado estiver no último nível da árvore, a operação de busca precisará realizar uma sequência de rotações zig-zig ou zig-zag para trazer o nó para

a raiz. Essa sequência de rotações pode ser longa e envolver vários níveis da árvore, o que pode levar a um tempo de execução menos eficiente.

Porém, é importante destacar que esse exemplo é o de uma situação específica e não representa necessariamente o pior caso em todas as circunstâncias. A árvore splay é projetada para otimizar o desempenho médio em uma ampla variedade de padrões de acesso, e seu desempenho em casos reais tende a ser eficiente na prática.

**Perda de balanceamento estrito:** As árvores splay não mantêm um balanceamento estrito após as operações. Embora isso não seja um problema na maioria das situações, em alguns casos, a estrutura pode se degradar e a eficiência das operações pode diminuir.

**Implementação mais complexa:** Embora sejam mais simples do que outras estruturas de dados balanceadas, as árvores splay requerem um cuidado especial na implementação das rotações e reorganizações. A complexidade do código pode ser maior em comparação com estruturas mais simples, como árvores binárias de busca tradicionais.

### Áreas de aplicação

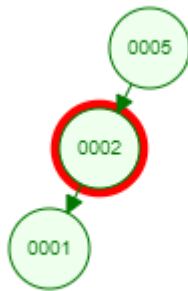
Existem diversos casos onde a árvore splay é o tipo mais rápido de árvore binária de busca. Sua capacidade de facilitar o acesso aos itens frequentemente acessados pode ser extremamente útil para a diminuição do tempo de acesso. Alguns exemplos de situações em que a aplicação de árvores splay é comum:

- **Cache:** Árvores splay são tipicamente utilizadas na implementação de sistemas de memória cache.
- **Indexação de Bancos de Dados:** Árvores splay podem ser utilizadas na indexação de bancos de dados para acelerar o processo de busca e recuperação dos dados.
- **Metadados do Sistema de Arquivos:** Árvores splay também podem ser usadas para armazenar de maneira mais eficiente metadados sobre o sistema de arquivos como a tabela de alocação, a estrutura das pastas e os atributos dos arquivos.

### Como é feita a Inclusão?

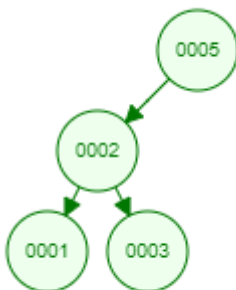
Primeiro, é necessário encaixar o elemento que desejamos inserir na posição correta. Após a inserção, é necessário trazer o nó inserido para a raiz da árvore. Esse processo é chamado de splay.

**1º passo:** encontrar a posição correta. (Considere que estamos inserindo o valor 3)



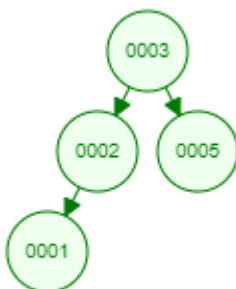
fonte: coletânea dos autores

**2º passo:** Inserir o elemento.



fonte: coletânea dos autores

**3º passo:** Realizar o splay.



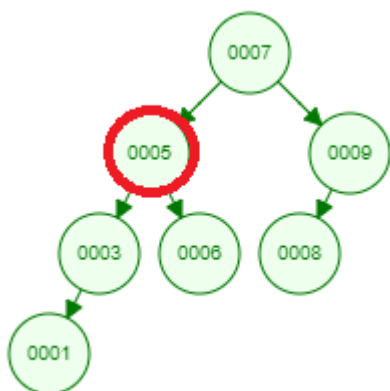
fonte: coletânea dos autores

### Como é feita a Exclusão?

Quando um nó é excluído de uma Splay Tree, três etapas são seguidas. Primeiro, precisa-se encontrar o nó que será excluído. Em seguida, esse nó será movido para o topo da árvore por meio de rotações. Por fim, será removido o nó da árvore. A exclusão em uma

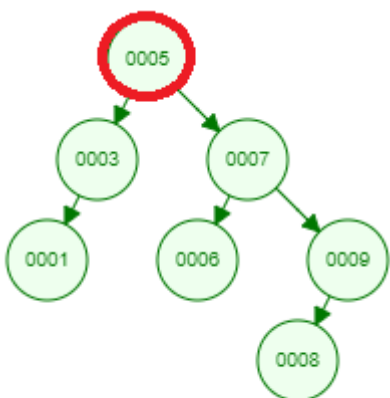
Splay Tree é um processo que equilibra a árvore e melhora a busca pelos nós mais usados. Vejamos na prática:

**1º passo:** encontrar o nó a ser deletado.



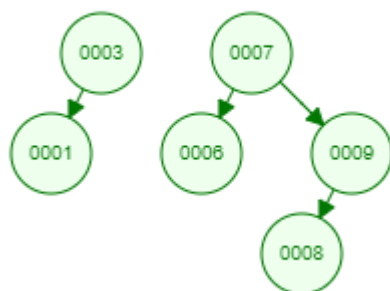
fonte: coletânea dos autores

**2º passo:** trazer o nó para a raiz da árvore. Aqui, por exemplo, foi feito o movimento Zig Right.



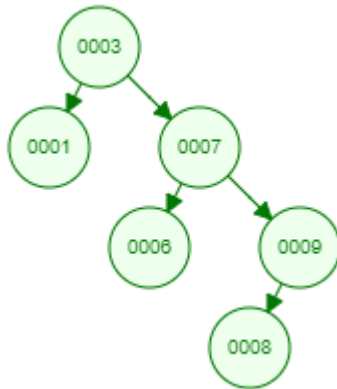
fonte: <https://www.cs.usfca.edu/~galles/visualization/SplayTree.html>

**3º passo:** reconectar a árvore.



fonte: coletânea dos autores

A preferência da raiz precisa ser definida e, neste caso, foi pelo maior valor à esquerda.



fonte: coletânea dos autores

### Exemplo de Código para Inclusão

```
INSERT(T, n,x)
```

```
// Criação de um novo nó e um nó temporário que recebe o endereço da raiz principal
```

```
node *y = NULL
```

```
node *temp = t -> root
```

```
// Enquanto a raiz temporária não for nula, (...)
```

```
while temp != NULL
```

```
// O novo nó recebe o nó temporário
```

```
y = temp
```

```
// Se o dado do nó, passado por parâmetro for menor que o dado do nó temporário, (...)
```

```
if n->data < temp->data
```

```
// Atribua o próximo nó à esquerda da variável temp
```

```
temp = temp->left
```

```
// Se não, atribua o próximo nó à direita da variável temp
```

```
temp = temp->right
```



**// Atribuindo um novo pai ao nó , (...)**

n->parent = y;

**// Se o novo nó for igual a nulo, (...)**

if y == NULL

**// A raiz da árvore será igual ao antigo nó**

t->root = n

**// Se não, se o dado do antigo nó for menor do que o novo nó**

**// O novo nó receberá o antigo nó como filho à esquerda**

y->left = n

**// Se não, o novo nó receberá o antigo nó como filho à direita**

y->right = n

**// Então, realize a operação de "splay" para posicionar o elemento y como raiz principal da árvore.**

SPLAY(t, n)

### **Exemplo de Código para Exclusão**

DELETE(T, n)

**// Criação de duas sub árvores splay, uma para a esquerda e outra para a direita**

left\_subtree = new splay\_tree

right\_subtree = new splay\_tree

**// Definição da raiz da subárvore esquerda como a raiz da árvore original, à esquerda**

left\_subtree.root = T.root.left

**// Definição da raiz da subárvore direita como a raiz da árvore original, à direita**

right\_subtree = T.root.right

**// Se a raiz da subárvore esquerda não for nula, (...)**

if left\_subtree.root != NULL

**// Defina o pai da raiz da subárvore esquerda como nulo**

left\_subtree.root.parent = NULL

```

// Se a raiz da subárvore direita não for nula, (...)
if right_subtree.root != NULL
    // Defina o pai da raiz da subárvore direita como nulo
    right_subtree.root.parent = NULL

// Se a raiz da subárvore esquerda não for nula, (...)
if left_subtree.root != NULL
    // Encontre o maior elemento na subárvore esquerda (left_subtree) e o coloca na
raiz (m)
    m = MAXIMUM(left_subtree, left_subtree.root)

    // Então, realize a operação de "splay" para posicionar o elemento m na raiz da
subárvore esquerda
    SPLAY(left_subtree, m)

    // Defina a raiz da subárvore esquerda como o filho direito da raiz da subárvore
direita
    left_subtree.root.right = right_subtree.root

    // Defina a raiz da árvore original (T) como a raiz da subárvore esquerda
    T.root = left_subtree.root
else
    // Se a raiz da subárvore esquerda for nula, a raiz da árvore original (T) será a raiz
da subárvore direita
    T.root = right_subtree.root

```

### Exemplo de Código para Busca

```

SEARCH(T, n, x)

//Se o valor buscado for igual a algum armazenado em nó
if x == n.data
    //Faça o splay do nó
    SPLAY(T, n)
    //E retorne o n
    return n

```

```
//Se não, caso seja menor que o nó buscado no momento
else if x < n.data
```

```
//Realize a busca pela esquerda
return search(T, n.left, x);
```

```
//Se não,
else if x > n.data
```

```
//Realize a busca pela direita
return search(T, n.right, x);
```

```
//Se não, retorne nulo -> Não encontrado!
else
return NULL
```

---

### Código geral da Splay Tree

→ Para melhor compreensão dos códigos de Incluir, Buscar e Deletar o nó da Árvore Splay, é proveitoso expôr o código geral da árvore, com atenção para fatores como tipagem, estruturação e organização de forma geral, por exemplo, e não apenas específica.

→ As figuras abaixo foram separadas em setores para melhor entendimento e visualização.


figura 1 - Tipagem



```
typedef struct node {
    int data;
    struct node *left;
    struct node *right;
    struct node *parent;
} node;
```

```
typedef struct splay_tree {
    struct node *root;
} splay_tree;
```


fonte: <https://www.codesdope.com/course/data-structures-splay-trees/>

**figura 2 - Novo nó**


```
node* new_node(int data) {
    node *n = malloc(sizeof(node));
    n->data = data;
    n->parent = NULL;
    n->right = NULL;
    n->left = NULL;

    return n;
}
```

fonte: <https://www.codesdope.com/course/data-structures-splay-trees/>


**figura 3 - Novo Splay Tree**


```
splay_tree* new_splay_tree() {
    splay_tree *t = malloc(sizeof(splay_tree));
    t->root = NULL;

    return t;
}
```

figura 4- Novo Splay

fonte: <https://www.codesdope.com/course/data-structures-splay-trees/>

**figura 4 - Nó Máximo**


```
node* maximum(splay_tree *t, node *x) {
    while(x->right != NULL)
        x = x->right;
    return x;
}
```

fonte: <https://www.codesdope.com/course/data-structures-splay-trees/>

**figura 5 - Rotação pela Esquerda**

```

void left_rotate(splay_tree *t, node *x) {
    node *y = x->right;
    x->right = y->left;

    if(y->left != NULL) {
        y->left->parent = x;
    }

    y->parent = x->parent;

    if(x->parent == NULL) { //x is root
        t->root = y;
    }

    else if(x == x->parent->left) { //x is left child
        x->parent->left = y;
    }

    else { //x is right child
        x->parent->right = y;
    }

    y->left = x;
    x->parent = y;
}

```

fonte: <https://www.codesdope.com/course/data-structures-splay-trees/>

**figura 6 - Rotação pela Direita**

```

void right_rotate(splay_tree *t, node *x) {
    node *y = x->left;
    x->left = y->right;

    if(y->right != NULL) {
        y->right->parent = x;
    }

    y->parent = x->parent;
    if(x->parent == NULL) { //x is root
        t->root = y;
    }

    else if(x == x->parent->right) { //x is left child
        x->parent->right = y;
    }

    else { //x is right child
        x->parent->left = y;
    }

    y->right = x;
    x->parent = y;
}

```

fonte: <https://www.codesdope.com/course/data-structures-splay-trees/>

figura 7 - Splay

```

void splay(splay_tree *t, node *n) {
    while(n->parent != NULL) { //node is not root

        if(n->parent == t->root) { //node is child of root, one rotation

            if(n == n->parent->left) {
                right_rotate(t, n->parent);
            }
            else {
                left_rotate(t, n->parent);
            }
        }
        else {
            node *p = n->parent;
            node *g = p->parent; //grandparent

            if(n->parent->left == n && p->parent->left == p) { //both are left children
                right_rotate(t, g);
                right_rotate(t, p);
            }

            else if(n->parent->right == n && p->parent->right == p) { //both are right children
                left_rotate(t, g);
                left_rotate(t, p);
            }

            else if(n->parent->right == n && p->parent->left == p) {
                left_rotate(t, p);
                right_rotate(t, g);
            }

            else if(n->parent->left == n && p->parent->right == p) {
                right_rotate(t, p);
                left_rotate(t, g);
            }
        }
    }
}

```

fonte: <https://www.codesdope.com/course/data-structures-splay-trees/>

figura 8 - Inserir

```

void insert(splay_tree *t, node *n) {
    node *y = NULL;
    node *temp = t->root;

    while(temp != NULL) {
        y = temp;

        if(n->data < temp->data)
            temp = temp->left;
        else
            temp = temp->right;
    }

    n->parent = y;

    if(y == NULL) //newly added node is root
        t->root = n;
    else if(n->data < y->data)
        y->left = n;
    else
        y->right = n;
}

```

fonte: <https://www.codesdope.com/course/data-structures-splay-trees/>

figura 9- Buscar

```

node* search(splay_tree *t, node *n, int x) {
    if(x == n->data) {
        splay(t, n);
        return n;
    }
    else if(x < n->data)
        return search(t, n->left, x);
    else if(x > n->data)
        return search(t, n->right, x);
    else
        return NULL;
}

```

fonte: <https://www.codesdope.com/course/data-structures-splay-trees/>

**figura 10 - Deletar**

```

void delete(splay_tree *t, node *n) {
    splay(t, n);

    splay_tree *left_subtree = new_splay_tree();
    left_subtree->root = t->root->left;

    if(left_subtree->root != NULL)
        left_subtree->root->parent = NULL;

    splay_tree *right_subtree = new_splay_tree();
    right_subtree->root = t->root->right;

    if(right_subtree->root != NULL)
        right_subtree->root->parent = NULL;

    free(n);

    if(left_subtree->root != NULL) {
        node *m = maximum(left_subtree, left_subtree->root);
        splay(left_subtree, m);
        left_subtree->root->right = right_subtree->root;
        t->root = left_subtree->root;
    }

    else {
        t->root = right_subtree->root;
    }
}

```

fonte: <https://www.codesdope.com/course/data-structures-splay-trees/>

**figura 11 - Ordenar**

```

void inorder(splay_tree *t, node *n) {
    if(n != NULL) {
        inorder(t, n->left);
        printf("%d\n", n->data);
        inorder(t, n->right);
    }
}

```

fonte: <https://www.codesdope.com/course/data-structures-splay-trees/>

## Bibliografia:

ACERVO LIMA, SPLAY TREE | CONJUNTO 1 (PESQUISA). Disponível em: <https://acervolima.com/splay-tree-conjunto-1-pesquisa/> Acesso em: 22 de junho de 2023

anikettchpiow, Introduction to Splay tree data structure. Disponível em: <https://www.geeksforgeeks.org/introduction-to-splay-tree-data-structure/> Acesso em: 22 de junho de 2023

Gaweda, A. Splay Trees - Intro [vídeo]. Canal do YouTube, 26 de junho de 2017. Disponível em: <https://www.youtube.com/watch?v=IMSt8upSqFk>. Acesso em: 22 de junho de 2023.

Gaweda, A. Splay Trees - Zig [vídeo]. Canal do YouTube, 26 de junho de 2017. Disponível em: [https://www.youtube.com/watch?v=46kPJS0B\\_mU](https://www.youtube.com/watch?v=46kPJS0B_mU). Acesso em: 22 de junho de 2023.

Gaweda, A. Splay Trees - Zig-Zag [vídeo]. Canal do YouTube, 26 de junho de 2017. Disponível em: <https://www.youtube.com/watch?v=tbP6INvKxng>. Acesso em: 22 de junho de 2023.

Jenny's Lectures CS IT, 5.19 Splay Tree Introduction | Data structure & Algorithm [vídeo] Canal do Youtube, 13 de novembro de 2019. Disponível em: <https://www.youtube.com/watch?v=qMmqOHR75b8> Acesso em: 22 de junho de 2023.

Pedrini, H. Estruturas de Dados. MC202. Instituto de Computação, UNICAMP. Disponível em: <http://www.ic.unicamp.br/~helio>. Acesso em: 22/06/2023

Simulador de Splay Tree. Disponível em: <https://www.cs.usfca.edu/~galles/visualization/SplayTree.html> Acesso em: 22 de junho de 2023.