Algoritmos e Estrutura de Dados II

Prof. Fellipe Guilherme Rey de Souza

Aula 19 – Árvore Splay

Agenda

- Introdução
- Espalhamento
- Quando Espalhar?
- Exemplos de Reestruturações
- Implementação

- **PS:** Parte do conteúdo retirado do material do Prof. Luiz Eduardo

Introdução

- A árvore Splay é diferente de todas as árvores balanceadas que foram apresentadas até aqui.

- Diferente das árvores balanceadas, a árvore Splay não mantém nenhuma informação de balanceamento (cor, inclinação ou fator de balanceamento) nos nós da árvore.
  - Ela não é uma árvore de balanceamento, mas sim uma **estrutura autoajustável**.

Introdução

- Na árvore Splay não está implementada nenhuma restrição explícita de balanceamento.
  - A única operação implementada é aquela que promove o nó consultado, inserido, ou removido para a raiz.
  - Esta operação é denominada **Splay** (espalhamento) que dá o nome para a estrutura.

Introdução

- O interessante é que esta operação permite garantir uma complexidade amortizada *O(log n)*, para a execução das operações de inserção, remoção e consulta na árvore Splay.

- A Árvore Splay, diferentemente das árvores AVL, Rubro-Negra, B e Heap, **também é modificada após a consulta** de um elemento.

Introdução

- Ou seja, após qualquer acesso (seja busca, inserção ou remoção), a Árvore Splay sofre rotações para mover o elemento acessado para a raiz.
  - Esse processo é chamado de splaying.

- Mesmo uma consulta (leitura ou busca) modifica a estrutura da árvore.
  - Isso é feito para que elementos acessados recentemente fiquem mais rápidos de acessar futuramente.
  - O que a caracteriza como uma estrutura autoajustável baseada em acesso!

Introdução

- A árvore Splay é usada principalmente em contextos onde o padrão de acesso aos dados é não uniforme, ou seja, alguns elementos são acessados com muito mais frequência que outros.

- Ela é útil quando se deseja melhorar o tempo de acesso médio sem a sobrecarga de manter o balanceamento rígido, como ocorre nas árvores AVL ou Rubro-Negra.

Introdução

- Ela é aplicada em estruturas onde o acesso recente implica maior chance de acesso futuro, como em caches, sistemas de arquivos, interpretadores de linguagens e compressores de dados.

- A árvore Splay também pode ser usada em implementações de tabelas de símbolos, listas de prioridade e em situações onde é importante garantir um bom desempenho amortizado das operações.

Espalhamento

- Dado um nó x de uma árvore Splay T, promove-se o nó x para a raiz de T através de uma série de reestruturações.

- Uma transformação específica é executada, dependendo da posição relativa de x, de y (o pai de x) e o z (o avô de x), se houver.

Espalhamento

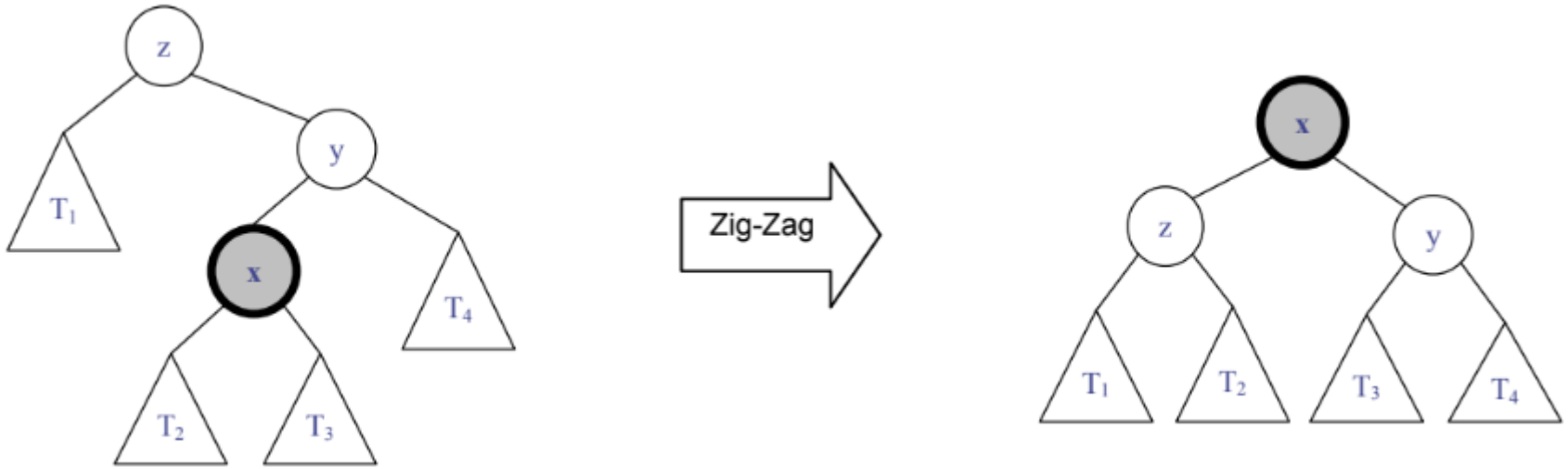
- Três operações de reestruturações são executadas:
  i. **Zig-Zag** (Duas rotações em lados opostos)
  ii. **Zig-Zig** (Duas rotações para o mesmo lado)
  iii. **Zig** (Uma rotação)

- A operação Splay (espalhamento) consiste em repetir as operações zig, zig-zig e zig-zag, sobre a chave x até que x torne-se a raiz da árvore T.

Espalhamento

- **Zig-Zag**
  - O nó *x* é o filho da esquerda (direita) e o nó *y* é o filho da direita (esquerda) de *z*.

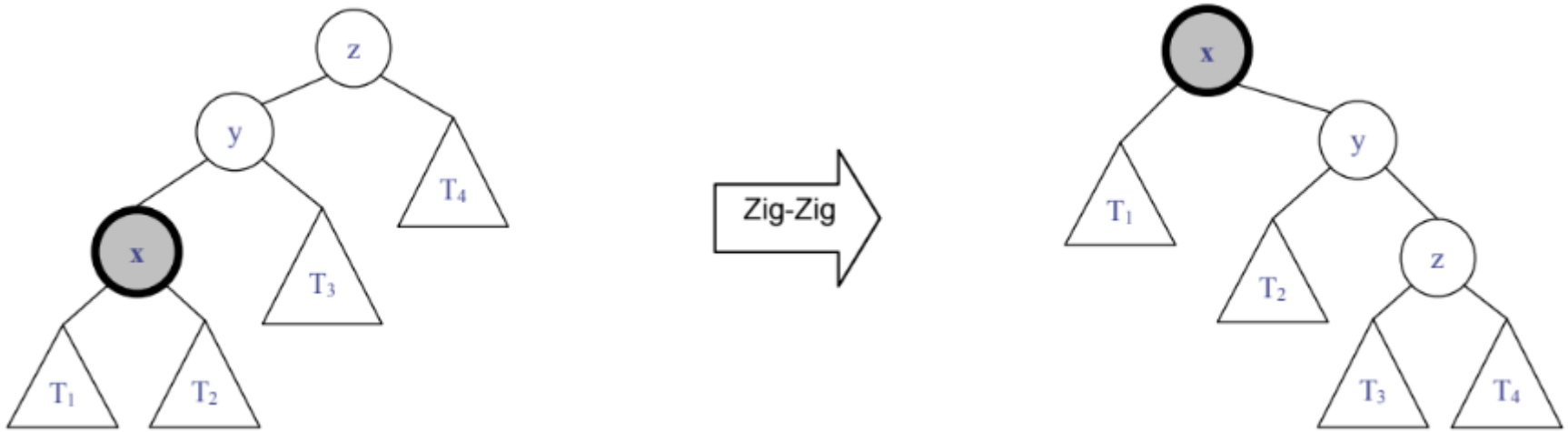
  Troca-se a árvore conforme a ilustração seguinte:



Espalhamento

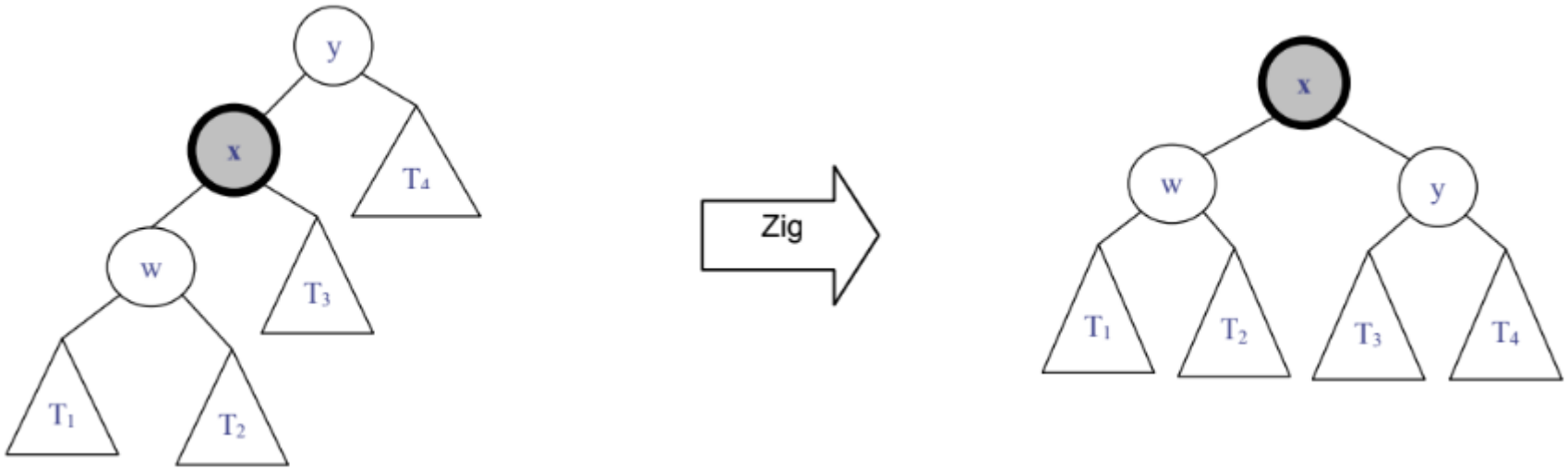
- **Zig-Zig**
  - O nó *x* é o filho da esquerda (direita) e o nó *y* é o filho da esquerda (direita) de *z*.

  Troca-se a árvore conforme a ilustração seguinte:



Espalhamento

- **Zig**
  - Não considera-se o avô de *x* nesta reestruturação. O nó *x* está a esquerda (direita) do nó *y*. A reestruturação é realizada conforme a ilustração seguinte:

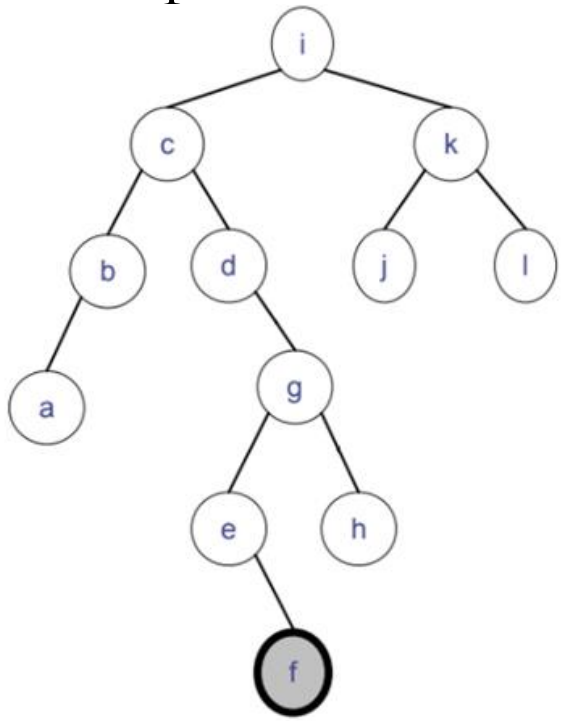


Quando espalhar?

- Quando pesquisamos, inserimos ou excluímos uma chave *k*, a reestruturação deve ser realizada, como segue:
  a. A pesquisa pela chave *k*, se a chave é encontrada ela é espalhada senão o pai do nó que deveria conter a chave *k* é que deve ser espalhada.
  b. A inserção da chave *k*, espalha-se o novo nó *k*.
  c. A remoção espalha-se o pai do nó *k* que é removido.

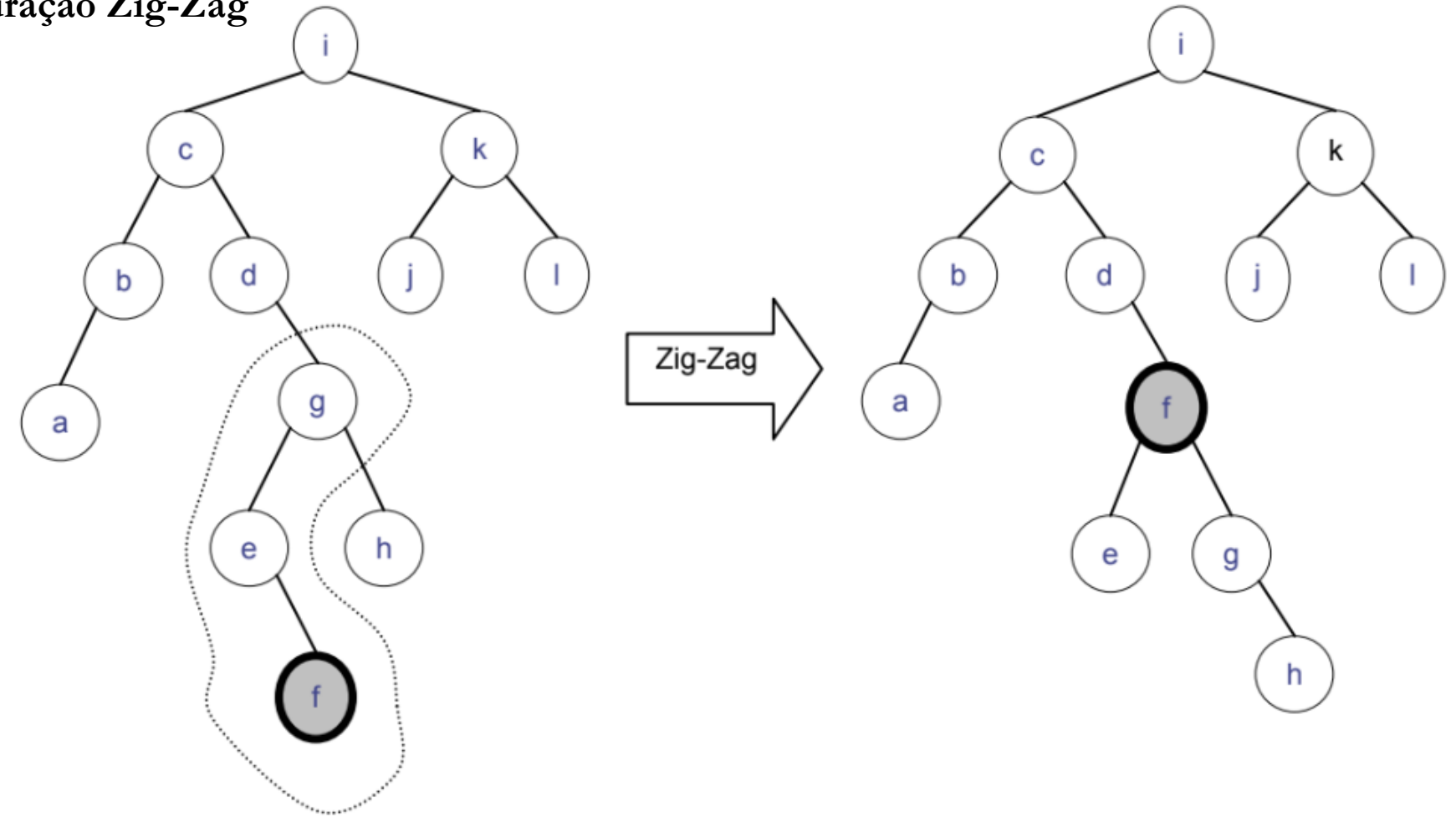
Quando espalhar?

- A seguir, será demonstrado uma sequência de reestruturações que deverão ser realizadas a fim de promover um nó (f) para a raiz da árvore.



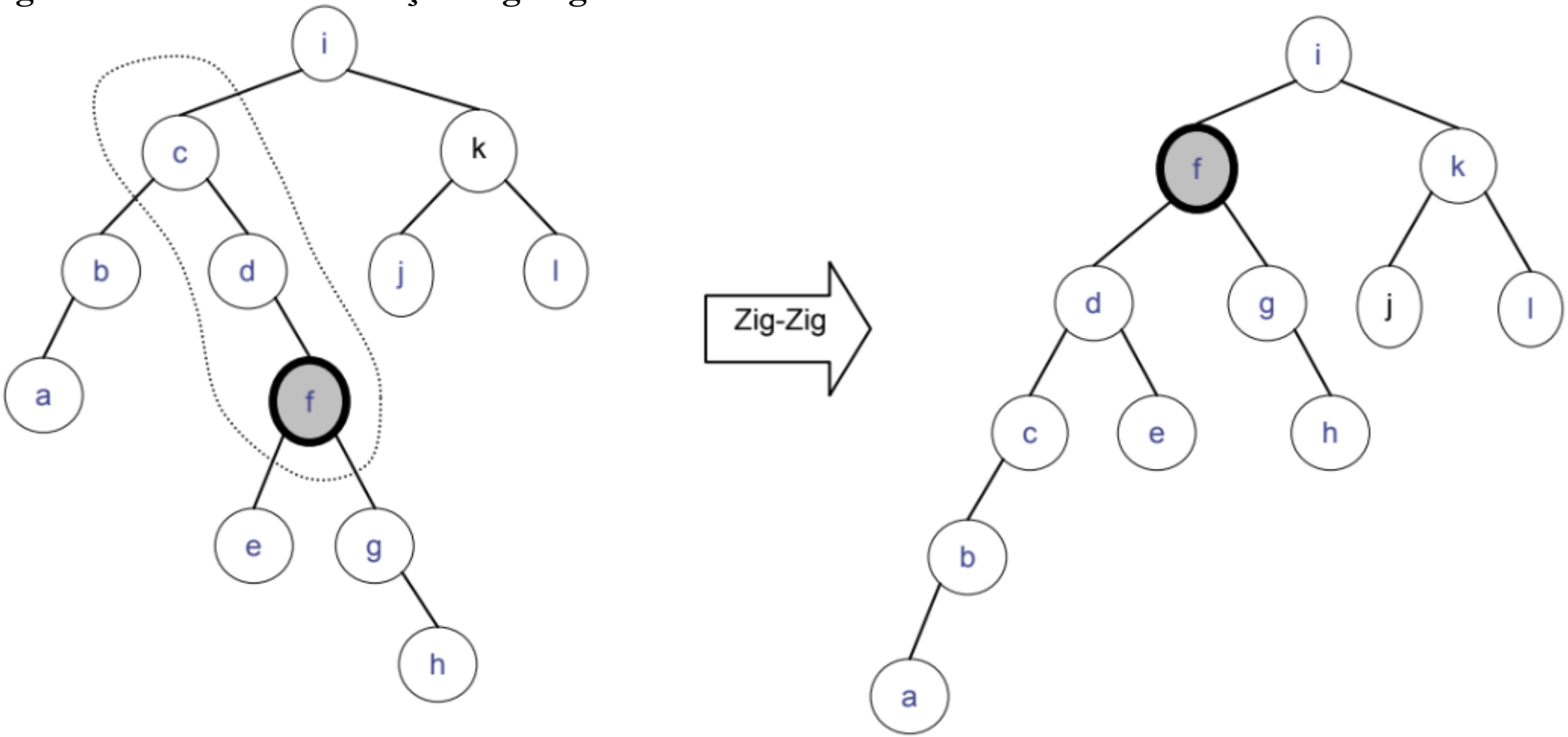
Quando espalhar?

(a) Reestruturação Zig-Zag



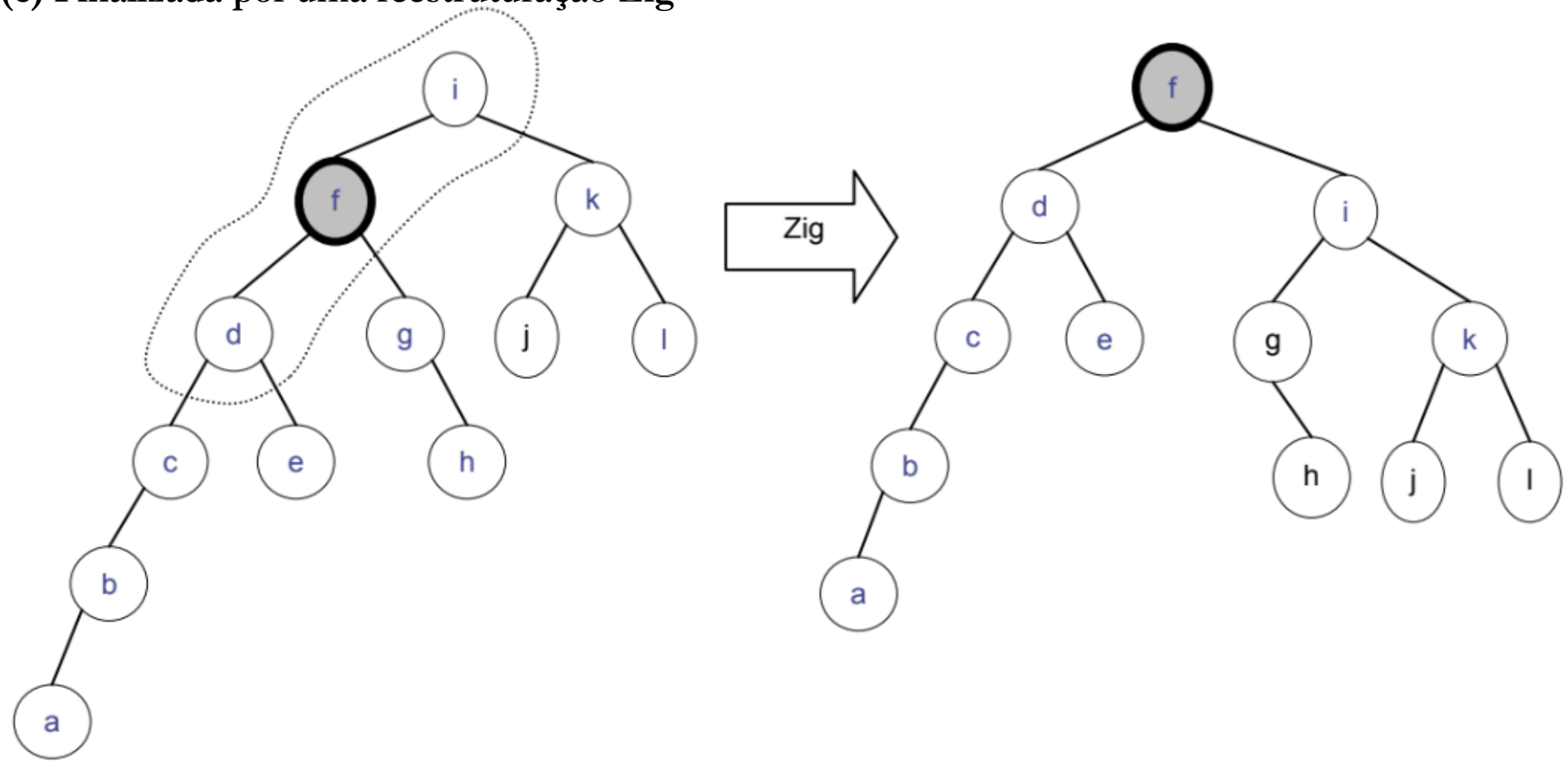
Quando espalhar?

(b) Seguida de uma reestruturação Zig-Zig

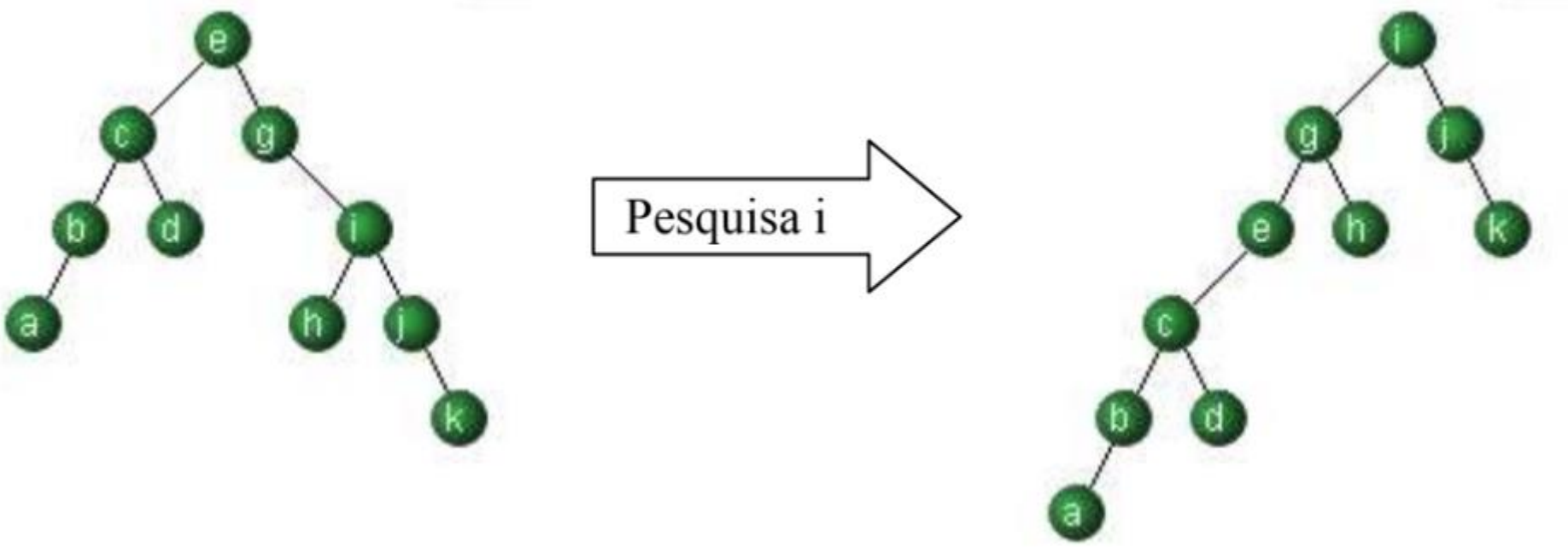


Quando espalhar?

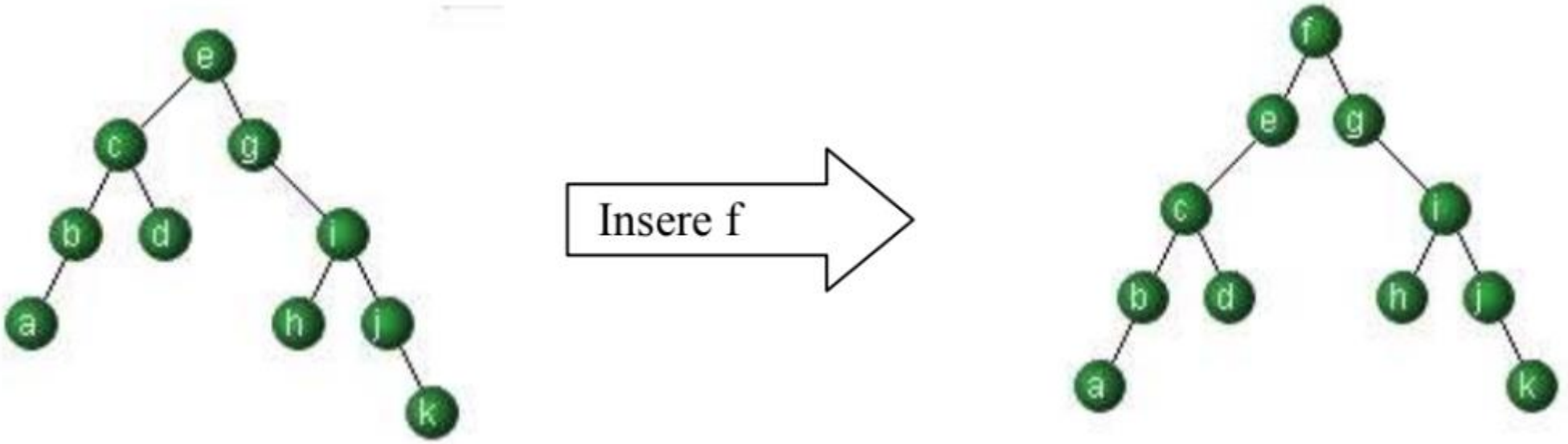
(c) Finalizada por uma reestruturação Zig



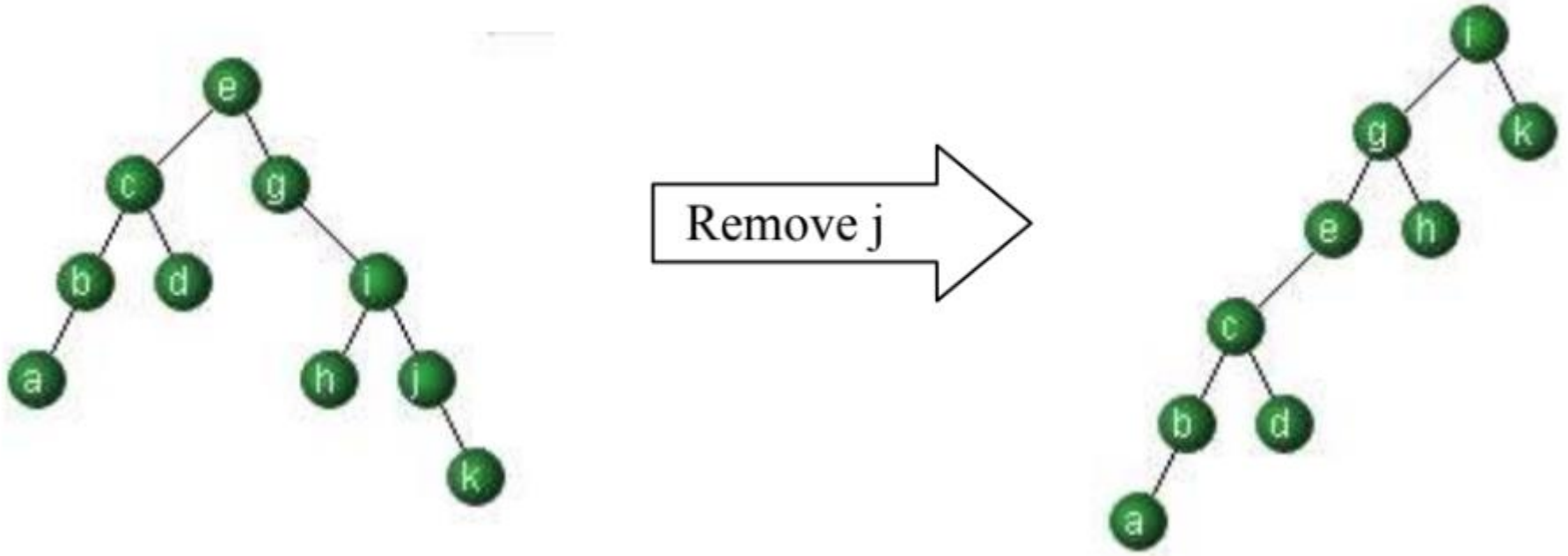
Exemplos de reestruturações



Exemplos de reestruturações



Exemplos de reestruturações



Implementação

```
typedef struct no {
    int info;
    struct no *esq;
    struct no *dir;
} Node;
```

- Para a Árvore Splay, vamos utilizar a mesma estrutura definida para a árvore binária de busca.

- Temos o conteúdo do nó da árvore, denominado *info*, e nós à esquerda (*esq*) e direita (*dir*) deste nó.

Implementação

- As funções de rotação direita e rotação esquerda (para utilizarmos as rotações Zig, Zig-Zag e Zig-Zig) são definidas exatamente igual à da Árvore AVL.

```
Node* rotacaoDireita(Node* x) {
    Node* y = x->esq;
    x->esq = y->dir;
    y->dir = x;
    return y;
}

Node* rotacaoEsquerda(Node* x) {
    Node* y = x->dir;
    x->dir = y->esq;
    y->esq = x;
    return y;
}
```

- A diferença está que na Árvore AVL nós chamávamos, ao fim da troca, a função *fatorBalanceamento* para o recálculo necessário.

Implementação

- A função splay tem o papel de mover a chave buscada para a raiz da árvore Splay, reorganizando os nós por meio de rotações.

- Ela começa verificando se a raiz é nula ou se já contém a chave desejada, caso em que não há mais nada a fazer e a raiz atual é retornada.

```
Node* splay(Node* raiz, int chave) {
    if (raiz == NULL || raiz->info == chave){
        return raiz;
    }
    if (chave < raiz->info) {
        if (raiz->esq == NULL) {
            return raiz;
        }
        if (chave < raiz->esq->info) {
            raiz->esq->esq = splay(raiz->esq->esq, chave);
            raiz = rotacaoDireita(raiz);
        } else if (chave > raiz->esq->info) {
            raiz->esq->dir = splay(raiz->esq->dir, chave);
            if (raiz->esq->dir != NULL)
                raiz->esq = rotacaoEsquerda(raiz->esq);
        }
        return (raiz->esq == NULL) ? raiz : rotacaoDireita(raiz);
    } else {
        if (raiz->dir == NULL) {
            return raiz;
        }
        if (chave > raiz->dir->info) {
            raiz->dir->dir = splay(raiz->dir->dir, chave);
            raiz = rotacaoEsquerda(raiz);
        } else if (chave < raiz->dir->info) {
            raiz->dir->esq = splay(raiz->dir->esq, chave);
            if (raiz->dir->esq != NULL)
                raiz->dir = rotacaoDireita(raiz->dir);
        }
        return (raiz->dir == NULL) ? raiz : rotacaoEsquerda(raiz);
    }
}
```

Implementação

- Se a chave procurada for menor que o valor da raiz, o algoritmo segue recursivamente para a subárvore esquerda.

- Se essa subárvore for nula, o elemento não está na árvore e a raiz atual é retornada.

```
Node* splay(Node* raiz, int chave) {
    if (raiz == NULL || raiz->info == chave){
        return raiz;
    }
    if (chave < raiz->info) {
        if (raiz->esq == NULL) {
            return raiz;
        }
        if (chave < raiz->esq->info) {
            raiz->esq->esq = splay(raiz->esq->esq, chave);
            raiz = rotacaoDireita(raiz);
        } else if (chave > raiz->esq->info) {
            raiz->esq->dir = splay(raiz->esq->dir, chave);
            if (raiz->esq->dir != NULL)
                raiz->esq = rotacaoEsquerda(raiz->esq);
        }
        return (raiz->esq == NULL) ? raiz : rotacaoDireita(raiz);
    } else {
        if (raiz->dir == NULL) {
            return raiz;
        }
        if (chave > raiz->dir->info) {
            raiz->dir->dir = splay(raiz->dir->dir, chave);
            raiz = rotacaoEsquerda(raiz);
        } else if (chave < raiz->dir->info) {
            raiz->dir->esq = splay(raiz->dir->esq, chave);
            if (raiz->dir->esq != NULL)
                raiz->dir = rotacaoDireita(raiz->dir);
        }
        return (raiz->dir == NULL) ? raiz : rotacaoEsquerda(raiz);
    }
}
```

Implementação

- Caso contrário, há duas situações possíveis. Se a chave também for menor que o valor do filho esquerdo (um caso chamado de zig-zig), ele aplica a função splay novamente na subárvore esquerda da subárvore esquerda, e depois faz uma rotação simples à direita na raiz atual.

```
Node* splay(Node* raiz, int chave) {
    if (raiz == NULL || raiz->info == chave){
        return raiz;
    }
    if (chave < raiz->info) {
        if (raiz->esq == NULL) {
            return raiz;
        }
        if (chave < raiz->esq->info) {
            raiz->esq->esq = splay(raiz->esq->esq, chave);
            raiz = rotacaoDireita(raiz);
        } else if (chave > raiz->esq->info) {
            raiz->esq->dir = splay(raiz->esq->dir, chave);
            if (raiz->esq->dir != NULL)
                raiz->esq = rotacaoEsquerda(raiz->esq);
        }
        return (raiz->esq == NULL) ? raiz : rotacaoDireita(raiz);
    } else {
        if (raiz->dir == NULL) {
            return raiz;
        }
        if (chave > raiz->dir->info) {
            raiz->dir->dir = splay(raiz->dir->dir, chave);
            raiz = rotacaoEsquerda(raiz);
        } else if (chave < raiz->dir->info) {
            raiz->dir->esq = splay(raiz->dir->esq, chave);
            if (raiz->dir->esq != NULL)
                raiz->dir = rotacaoDireita(raiz->dir);
        }
        return (raiz->dir == NULL) ? raiz : rotacaoEsquerda(raiz);
    }
}
```

Implementação

- Se a chave for maior que o valor do filho esquerdo (caso zig-zag), a função é aplicada na subárvore direita da subárvore esquerda.

- Se essa subárvore não for nula, é feita uma rotação à esquerda no filho esquerdo, preparando para uma rotação à direita logo em seguida.

```
Node* splay(Node* raiz, int chave) {
    if (raiz == NULL || raiz->info == chave){
        return raiz;
    }
    if (chave < raiz->info) {
        if (raiz->esq == NULL) {
            return raiz;
        }
        if (chave < raiz->esq->info) {
            raiz->esq->esq = splay(raiz->esq->esq, chave);
            raiz = rotacaoDireita(raiz);
        } else if (chave > raiz->esq->info) {
            raiz->esq->dir = splay(raiz->esq->dir, chave);
            if (raiz->esq->dir != NULL)
                raiz->esq = rotacaoEsquerda(raiz->esq);
        }
        return (raiz->esq == NULL) ? raiz : rotacaoDireita(raiz);
    } else {
        if (raiz->dir == NULL) {
            return raiz;
        }
        if (chave > raiz->dir->info) {
            raiz->dir->dir = splay(raiz->dir->dir, chave);
            raiz = rotacaoEsquerda(raiz);
        } else if (chave < raiz->dir->info) {
            raiz->dir->esq = splay(raiz->dir->esq, chave);
            if (raiz->dir->esq != NULL)
                raiz->dir = rotacaoDireita(raiz->dir);
        }
        return (raiz->dir == NULL) ? raiz : rotacaoEsquerda(raiz);
    }
}
```

Implementação

- Se a chave for maior que a raiz, a lógica segue de forma simétrica na subárvore direita. Primeiro, verifica se a subárvore direita existe. Se não, a raiz é retornada.

- Caso exista, há também dois casos. Se a chave for maior que o valor do filho direito (zig-zig), a função é aplicada recursivamente na subárvore direita da subárvore direita e uma rotação à esquerda é realizada.

```
Node* splay(Node* raiz, int chave) {
    if (raiz == NULL || raiz->info == chave){
        return raiz;
    }
    if (chave < raiz->info) {
        if (raiz->esq == NULL) {
            return raiz;
        }
        if (chave < raiz->esq->info) {
            raiz->esq->esq = splay(raiz->esq->esq, chave);
            raiz = rotacaoDireita(raiz);
        } else if (chave > raiz->esq->info) {
            raiz->esq->dir = splay(raiz->esq->dir, chave);
            if (raiz->esq->dir != NULL)
                raiz->esq = rotacaoEsquerda(raiz->esq);
        }
        return (raiz->esq == NULL) ? raiz : rotacaoDireita(raiz);
    } else {
        if (raiz->dir == NULL) {
            return raiz;
        }
        if (chave > raiz->dir->info) {
            raiz->dir->dir = splay(raiz->dir->dir, chave);
            raiz = rotacaoEsquerda(raiz);
        } else if (chave < raiz->dir->info) {
            raiz->dir->esq = splay(raiz->dir->esq, chave);
            if (raiz->dir->esq != NULL)
                raiz->dir = rotacaoDireita(raiz->dir);
        }
        return (raiz->dir == NULL) ? raiz : rotacaoEsquerda(raiz);
    }
}
```

Implementação

- Se a chave for menor que o filho direito (zig-zag), a função é aplicada na subárvore esquerda da subárvore direita.

- Se essa subárvore não for nula, realiza-se uma rotação à direita no filho direito, preparando para uma rotação à esquerda na raiz.

```
Node* splay(Node* raiz, int chave) {
    if (raiz == NULL || raiz->info == chave){
        return raiz;
    }
    if (chave < raiz->info) {
        if (raiz->esq == NULL) {
            return raiz;
        }
        if (chave < raiz->esq->info) {
            raiz->esq->esq = splay(raiz->esq->esq, chave);
            raiz = rotacaoDireita(raiz);
        } else if (chave > raiz->esq->info) {
            raiz->esq->dir = splay(raiz->esq->dir, chave);
            if (raiz->esq->dir != NULL)
                raiz->esq = rotacaoEsquerda(raiz->esq);
        }
        return (raiz->esq == NULL) ? raiz : rotacaoDireita(raiz);
    } else {
        if (raiz->dir == NULL) {
            return raiz;
        }
        if (chave > raiz->dir->info) {
            raiz->dir->dir = splay(raiz->dir->dir, chave);
            raiz = rotacaoEsquerda(raiz);
        } else if (chave < raiz->dir->info) {
            raiz->dir->esq = splay(raiz->dir->esq, chave);
            if (raiz->dir->esq != NULL)
                raiz->dir = rotacaoDireita(raiz->dir);
        }
        return (raiz->dir == NULL) ? raiz : rotacaoEsquerda(raiz);
    }
}
```

Implementação

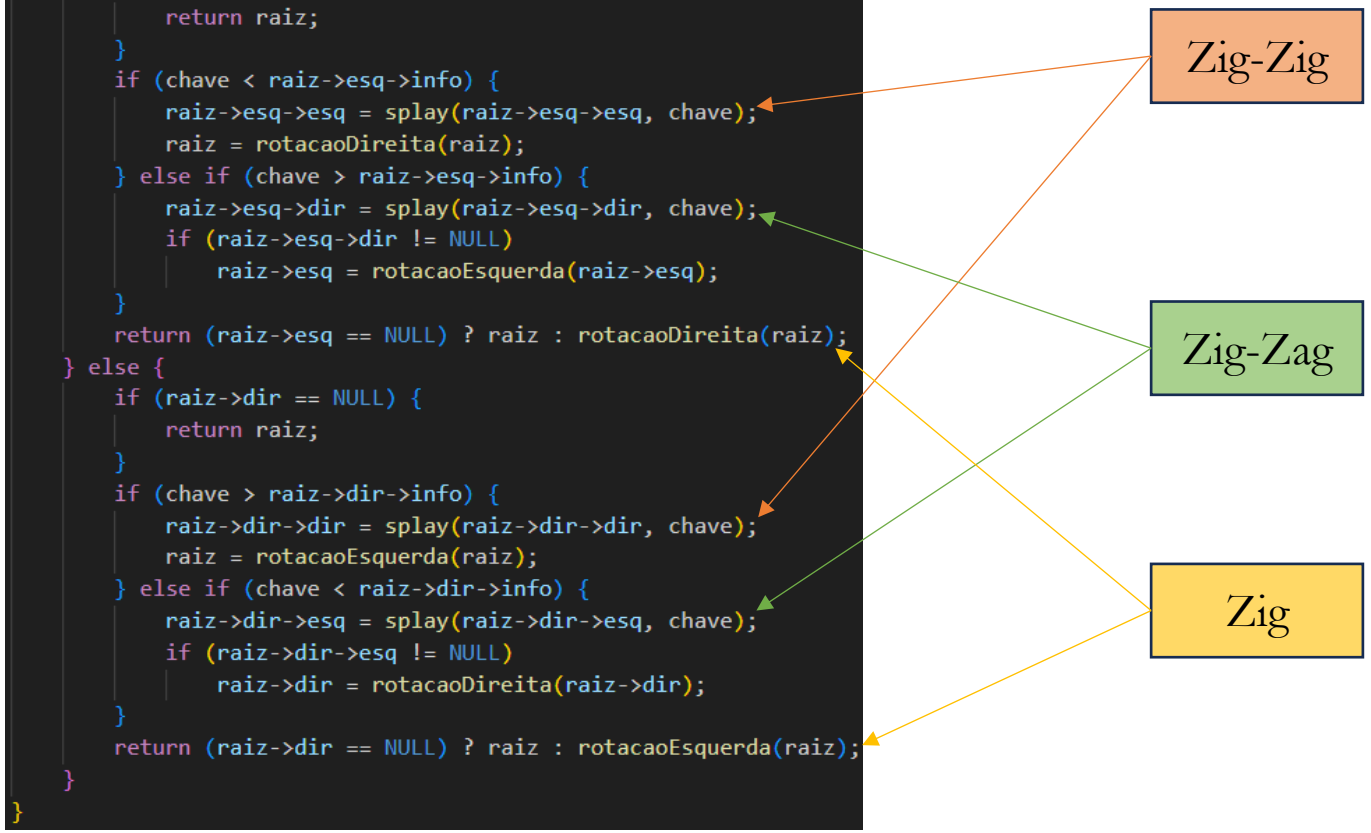
- No final de cada ramo da função, o retorno será sempre a nova raiz da árvore resultante da reorganização, com o objetivo de aproximar a chave buscada ao topo.

- O efeito geral é trazer a chave acessada para a raiz da árvore, mantendo as propriedades de árvore binária de busca e melhorando o desempenho de acessos futuros.

```
Node* splay(Node* raiz, int chave) {
    if (raiz == NULL || raiz->info == chave){
        return raiz;
    }
    if (chave < raiz->info) {
        if (raiz->esq == NULL) {
            return raiz;
        }
        if (chave < raiz->esq->info) {
            raiz->esq->esq = splay(raiz->esq->esq, chave);
            raiz = rotacaoDireita(raiz);
        } else if (chave > raiz->esq->info) {
            raiz->esq->dir = splay(raiz->esq->dir, chave);
            if (raiz->esq->dir != NULL)
                raiz->esq = rotacaoEsquerda(raiz->esq);
        }
        return (raiz->esq == NULL) ? raiz : rotacaoDireita(raiz);
    } else {
        if (raiz->dir == NULL) {
            return raiz;
        }
        if (chave > raiz->dir->info) {
            raiz->dir->dir = splay(raiz->dir->dir, chave);
            raiz = rotacaoEsquerda(raiz);
        } else if (chave < raiz->dir->info) {
            raiz->dir->esq = splay(raiz->dir->esq, chave);
            if (raiz->dir->esq != NULL)
                raiz->dir = rotacaoDireita(raiz->dir);
        }
        return (raiz->dir == NULL) ? raiz : rotacaoEsquerda(raiz);
    }
}
```

Implementação


```
Node* splay(Node* raiz, int chave) {
    if (raiz == NULL || raiz->info == chave){
        return raiz;
    }
    if (chave < raiz->info) {
        if (raiz->esq == NULL) {
            return raiz;
        }
        if (chave < raiz->esq->info) {
            raiz->esq->esq = splay(raiz->esq->esq, chave);
            raiz = rotacaoDireita(raiz);
        } else if (chave > raiz->esq->info) {
            raiz->esq->dir = splay(raiz->esq->dir, chave);
            if (raiz->esq->dir != NULL)
                raiz->esq = rotacaoEsquerda(raiz->esq);
        }
        return (raiz->esq == NULL) ? raiz : rotacaoDireita(raiz);
    } else {
        if (raiz->dir == NULL) {
            return raiz;
        }
        if (chave > raiz->dir->info) {
            raiz->dir->dir = splay(raiz->dir->dir, chave);
            raiz = rotacaoEsquerda(raiz);
        } else if (chave < raiz->dir->info) {
            raiz->dir->esq = splay(raiz->dir->esq, chave);
            if (raiz->dir->esq != NULL)
                raiz->dir = rotacaoDireita(raiz->dir);
        }
        return (raiz->dir == NULL) ? raiz : rotacaoEsquerda(raiz);
    }
}
```



Algoritmos e Estrutura de Dados II

Prof. Fellipe Guilherme Rey de Souza

Aula 19 – Árvore Splay