Estruturas de Dados 2

aula 05 – Árvore Rubro-Negra – parte 1

Antonio Angelo de Souza Tartaglia

angelot@ifsp.edu.br

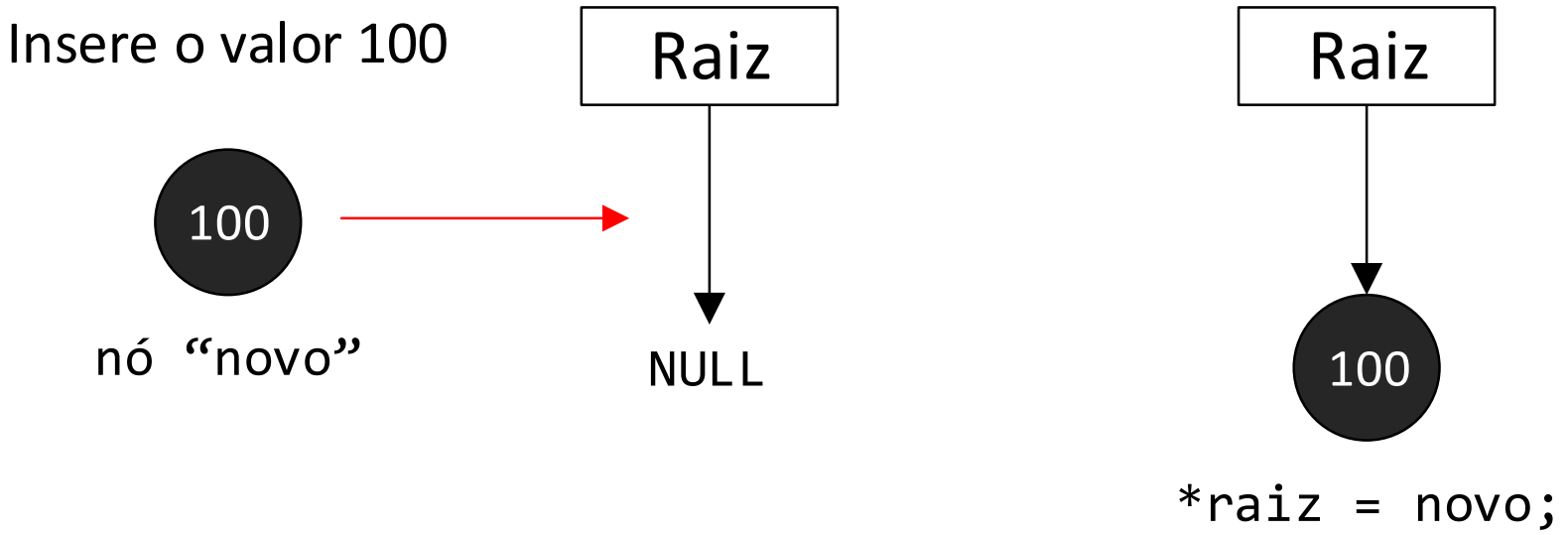
Inserção na Árvore Rubro-Negra caída para a esquerda – LLRB

- Para inserir um valor “V” na Árvore Rubro-Negra:
  - Se a raiz é NULL: insira o nó;
  - Se “V” é menor do que a raiz: vá para a Sub-Árvore da esquerda;
  - Se “V” é maior do que a raiz: vá para a Sub-Árvore da direita;
  - Aplique o método recursivamente.

- Ao voltar da recursão, verifique as propriedades de cada Sub-Árvore;
- Aplique a rotação ou mudança de cor necessária se alguma propriedade foi violada

Inserção na Árvore Rubro-Negra caída para a esquerda – LLRB

- Caso onde a inserção é feita em uma Árvore Rubro-Negra que está vazia



Inserção na Árvore Rubro-Negra caída para a esquerda – LLRB

```c
//Arquivo arvoreLLRB.h
int insere_arvoreLLRB(arvoreLLRB *raiz, int valor);
```

```c
int insere_arvoreLLRB(arvoreLLRB *raiz, int valor){
    int resp;
    //função responsável pela busca do local de inserção do nó
    *raiz = insereNO(*raiz, valor, &resp);
    if((*raiz) != NULL){
        (*raiz)->cor = BLACK;
    }
    return resp;
}
```

```c
//programa principal
x = insere_arvoreLLRB(raiz, 150);
x = insere_arvoreLLRB(raiz, 110);
x = insere_arvoreLLRB(raiz, 100);
x = insere_arvoreLLRB(raiz, 130);
x = insere_arvoreLLRB(raiz, 120);
x = insere_arvoreLLRB(raiz, 140);
x = insere_arvoreLLRB(raiz, 160);
```

Inserção na Árvore Rubro-Negra caída para a esquerda – LLRB

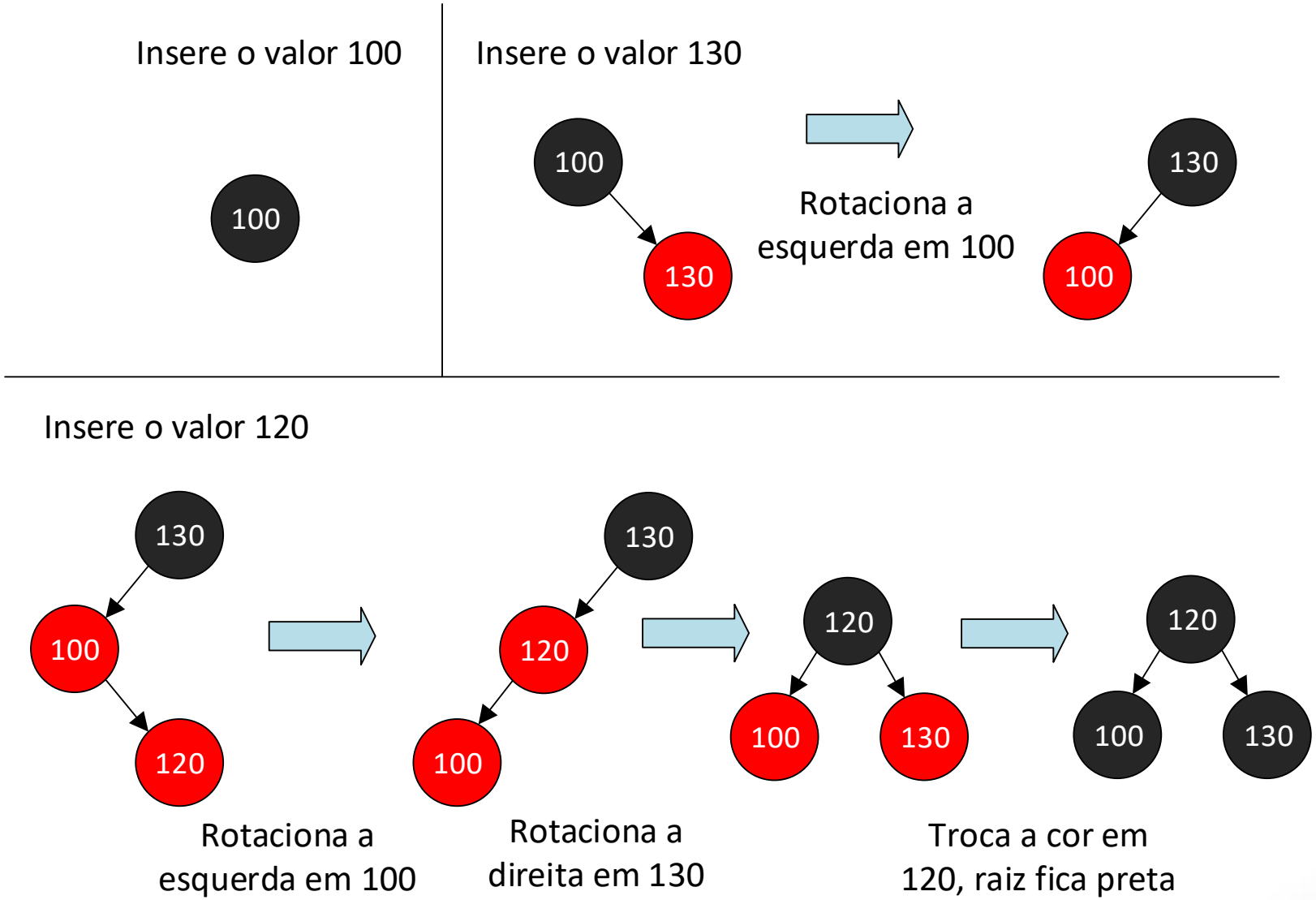
```
struct NO *insereNO(struct NO *H, int valor, int *resp){
    if(H == NULL){
        struct NO *novo;
        novo = (struct NO*) malloc(sizeof(struct NO));
        if(novo == NULL){
            *resp = 0;
            return NULL;
        }
        novo->info = valor;
        novo->cor  = RED;
        novo->dir  = NULL;
        novo->esq  = NULL;
        *resp = 1;
        return novo;
    }
    if(valor == H->info){
        *resp = 0;
    }else{
        if(valor < H->info){
            H->esq = insereNO(H->esq, valor, resp);
        }else{
            H->dir = insereNO(H->dir, valor, resp);
        }
    }
    if(cor(H->dir) == RED && cor(H->esq) == BLACK){
        H = rotacionaEsquerda(H);
    }
    if(cor(H->esq) == RED && cor(H->esq->esq) == RED){
        H = rotacionaDireita(H);
    }
    if(cor(H->esq) == RED && cor(H->dir) == RED){
        trocaCor(H);
    }
    return H;
}
```

Execução das chamadas recursivas, descendo na árvore para a direita ou esquerda

Insere nó sempre como folha, vermelho, e sempre tratando H como raiz

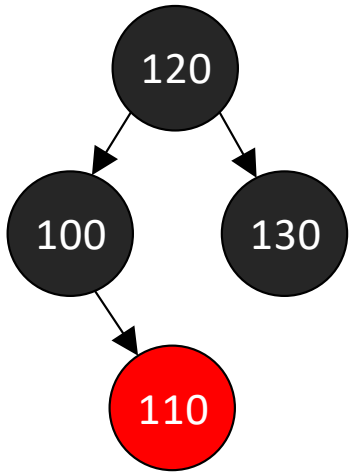
Acerta o balanceamento na volta de cada recursão

Inserção na Árvore Rubro-Negra caída para a esquerda – LLRB

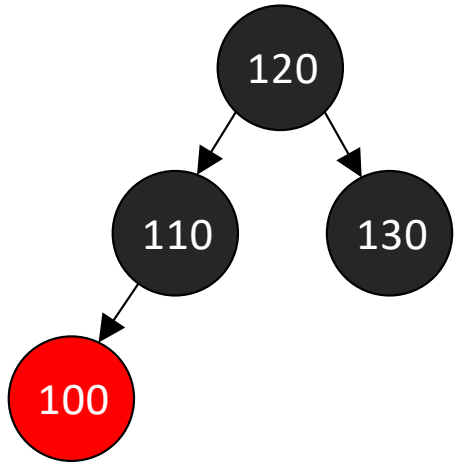


Inserção na Árvore Rubro-Negra caída para a esquerda – LLRB

Insere o valor 110



Rotaciona a esquerda em 100



Remoção na Árvore Rubro-Negra caída para a esquerda – LLRB

- Existem 3 tipos de remoção:
  - Nó folha (sem filhos);
  - Nó folha com 1 filho;
  - Nó folha com 2 filhos.

- Os três tipos de remoção trabalham juntos. A remoção sempre remove um elemento específico da Árvore, o qual pode ser um nó folha, ter um ou dois filhos;

Remoção na Árvore Rubro-Negra caída para a esquerda – LLRB

- Cuidado:
  - Não se pode remover de uma Árvore vazia;
  - Removendo o último nó, a Árvore fica vazia;

- Balanceamento:
  - Ao voltar na recursão, verifique as propriedades de cada Sub-Árvore;
  - Aplique a rotação ou mudança de cor necessária, se alguma propriedade foi violada.

Remoção na Árvore Rubro-Negra caída para a esquerda – LLRB

```c
//Arquivo arvoreLLRB.h
int remove_arvoreLLRB(arvoreLLRB *raiz, int valor);

//programa principal
remove_arvoreLLRB(raiz, 100);

int remove_arvoreLLRB(arvoreLLRB *raiz, int valor){
    if(consulta_arvoreLLRB(raiz, valor)){
        struct NO *H = *raiz;
        //função responsável pela busca pelo nó a ser removido
        *raiz = removeNO(H, valor);
        if(*raiz != NULL){
            (*raiz)->cor = BLACK;
        }
        return 1;
    }else{
        return 0;
    }
}
```

Esta função é de gerenciamento e garante que a raiz sempre seja preta. Existindo o nó a ser removido, dispara outra função que é efetivamente a responsável pela remoção.

A função de remoção não é tão simples quanto a inserção. A remoção parte do principio que o nó que se quer remover existe, por isso, a medida que a função vai descendo na árvore, ela já vai reestruturando-a. Para garantir que o nó exista, antes é necessário uma consulta. Se a consulta retornar verdadeiro, então o nó pode ser removido, senão, retorna 0

Estrutura de Dados 2

```
struct NO *removeNO(struct NO *H, int valor){
    if(valor < H->info){
        if(cor(H->esq) == BLACK && cor(H->esq->esq) == BLACK){
            H = move2EsqRED(H);
        }
        H->esq = removeNO(H->esq, valor);
    }else{
        if(cor(H->esq) == RED){
            H = rotacionaDireita(H);
        }
        if(valor == H->info && (H->dir == NULL)){
            free(H);
            return NULL;
        }
        if(cor(H->dir) == BLACK && cor(H->dir->esq) == BLACK){
            H = move2DirRED(H);
        }
        if(valor == H->info){
            struct NO *x = procuraMenor(H->dir);
            H->info = x->info;
            H->dir = removeMenor(H->dir);
        }else{
            H->dir = removeNO(H->dir, valor);
        }
    }
    return balancear(H);
}
```

Na busca a quem remover, já desce balanceando

Se valor procurado for menor do que H->info, move H para a esquerda e chama a função remove nó para a sub-árvore da esquerda.

Caso em que H é um nó folha e não possui filhos

Troca a informação do nó H pela informação do nó x, em seguida procura o menor nó novamente (que é o x), e o remove.

Procura o menor na Sub-Árvore da direita. É igual ao que é feito na Árvore AVL. Não pode simplesmente remover o nó, é necessário substituí-lo

Antes de voltar a recursão, chama a função balancear, que considera os três casos de violação das propriedades da Árvore

Remoção na Árvore Rubro-Negra caída para a esquerda – LLRB

```
struct NO *removeMenor(struct NO *H){
    if(H->esq == NULL){
        free(H);
        return NULL;
    }
    if(cor(H->esq) == BLACK && cor(H->esq->esq) == BLACK){
        H = move2EsqRED(H);
    }
    H->esq = removeMenor(H->esq);
    return balancear(H);
}
```

Se não tiver filho da esquerda, é só apagar H e retornar

Se tiver filho da esquerda, trata possíveis problemas de balanceamento

Chama recursivamente a mesma função. Em algum momento a condição do 1º if será satisfeita, sempre procurando à esquerda.

Quando estiver voltando da recursão, trata o balanceamento.

Remoção na Árvore Rubro-Negra caída para a esquerda – LLRB

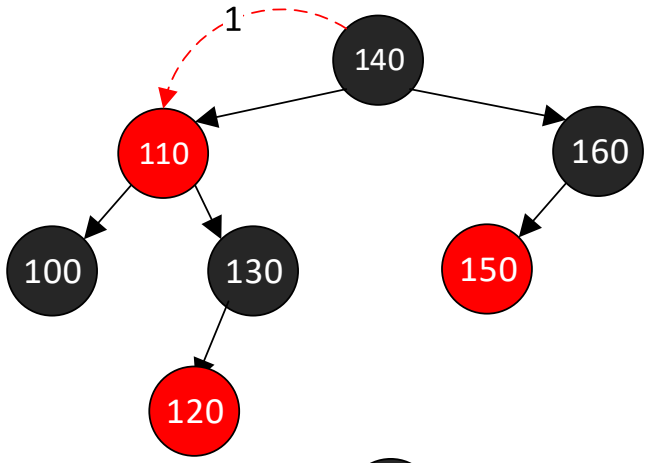
```
struct NO *procuraMenor(struct NO *atual){
    struct NO *no1 = atual;
    struct NO *no2 = atual->esq;
    while(no2 != NULL){
        no1 = no2;
        no2 = no2->esq;
    }
    return no1;
}
```

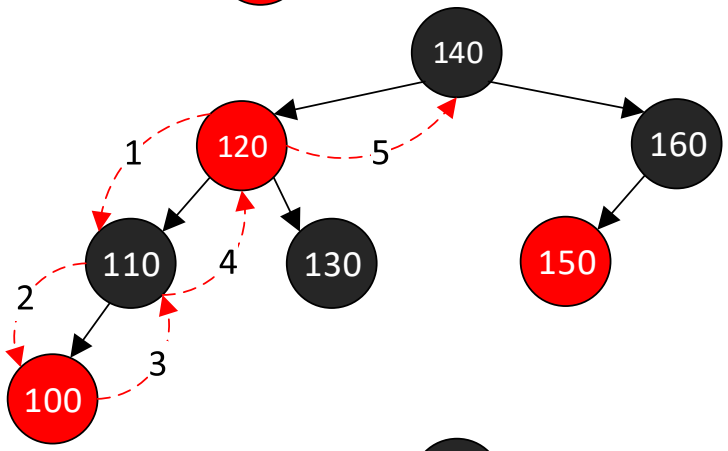
Procura pelo nó mais a esquerda

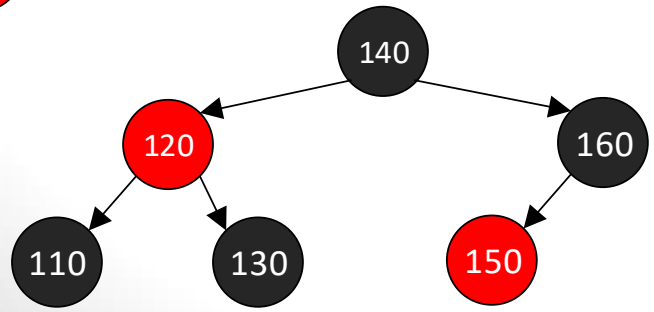
Os nós se movem cada vez mais a esquerda até chegar em NULL

Estrutura de Dados 2

Remove o valor 100







Inicia a busca pelo nó a ser removido a partir do 140

Nó procurado é menor do que 140. Visita nó 110.

Nó 110 tem filho e neto (NULL) da cor preta à esquerda.
Chama a função `move2esqRED( )`

Continua a busca a partir do nó 120.

1 – Nó procurado é menor do que 120. Visita nó 110.

2 – Nó procurado é menor do que 110. Visita nó 100.

3 – Nó a ser removido foi encontrado. Libera nó e volta para nó 110.

4 – Balanceamento no nó 110 está OK. Volta para nó 120.

5 – Balanceamento no nó 120 está OK. Volta para nó 140.

Balanceamento no nó 140 está OK. Processo de remoção termina.

Árvore Rubro-Negra caída para a esquerda – LLRB

- Entregue a atividade árvore Rubro-Negra completa no Moodle.