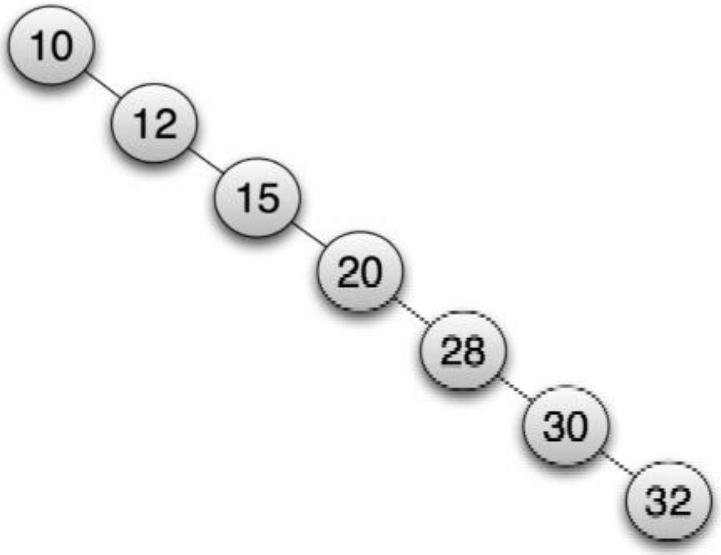
ESTRUTURAS DE DADOS II – ÁRVORE BALANCEADA

Prof. Patrícia Noll de Mattos

Árvores Balanceadas

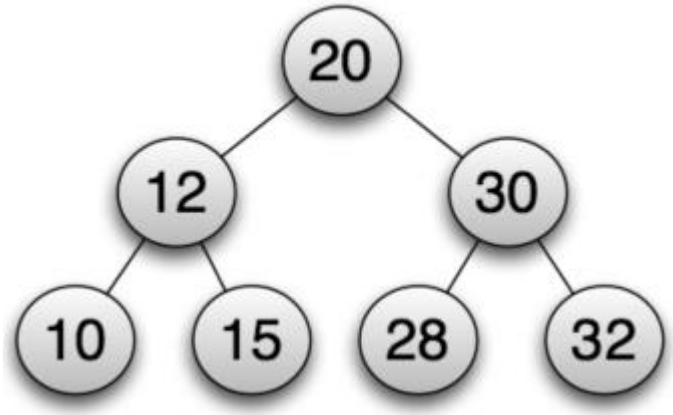
- As buscas em uma árvore binária são otimizadas se seus elementos estão bem distribuídos na árvore.
- Imagine a árvore a seguir, com os elementos inseridos nesta ordem: 10, 12, 15, 20, 28, 10, 32:
- Busca pelo elemento 32...



Para encontrar o elemento 32, são necessárias 7 comparações.

Árvore Balanceada

- Distância média entre os nodos e a raiz é mínima:
- **Árvore exemplo:**
  - **Mesmo número de elementos** na subárvore esquerda e direita.
  - **É uma árvore binária completa:** todos os nodos folha encontram-se no último nível da árvore.



Para encontrar o elemento 32, são necessárias 3 comparações.

Condição para Balanceamento

- A altura da subárvore esquerda deve diferir da altura da subárvore direita em no máximo **1** (-1, 0, 1).

- **Nodos AVL:** possuem um atributo a mais, a **altura da subárvore** cujo nodo é raiz.

- **Fator de balanceamento** (ou fator do nodo):
- **Fator(n) = h(esq(n)) – h(dir(n)),** sendo:
- **h** função que **retorna altura** do nodo;
- **esq** – função que **retorna a raiz** da subárvore **esquerda** do nodo.
- **dir** – função que **retorna a raiz** da subárvore **direita** do nodo.

Nodos AVL (Árvore Balanceada)

```
struct nodo{
        int alt;
        int chave;
        struct nodo *esquerda, *direita;
};

struct nodo *crianodo(int el) {
                struct nodo *p=NULL;
                p=(struct nodo *) malloc(sizeof(struct nodo));
                if (p!=NULL){
                        p->alt=0;
                        p->chave=el;
                        p->esquerda = NULL;
                        p->direita = NULL;
                }
                return p;
}
```

Nodos AVL

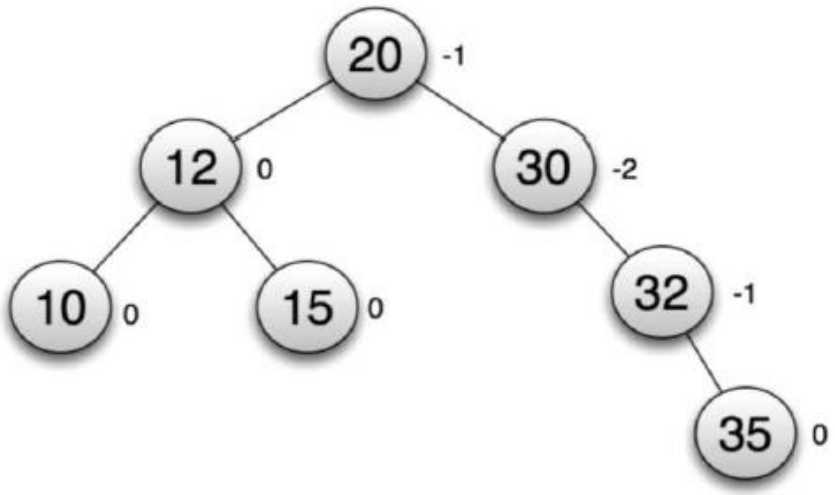
```
int altura(struct nodo *raiz){
        if(raiz==NULL) return 0;
        else return raiz->alt;
}

int maximo(int a1, int a2){
        if(a1>a2) return a1;
        else return a2;
}

int getfator(struct nodo *raiz){
        struct nodo *p=raiz;
        if(p!=NULL)
        return altura(p->esquerda) - altura(p->direita);
        else return 0;
}
```

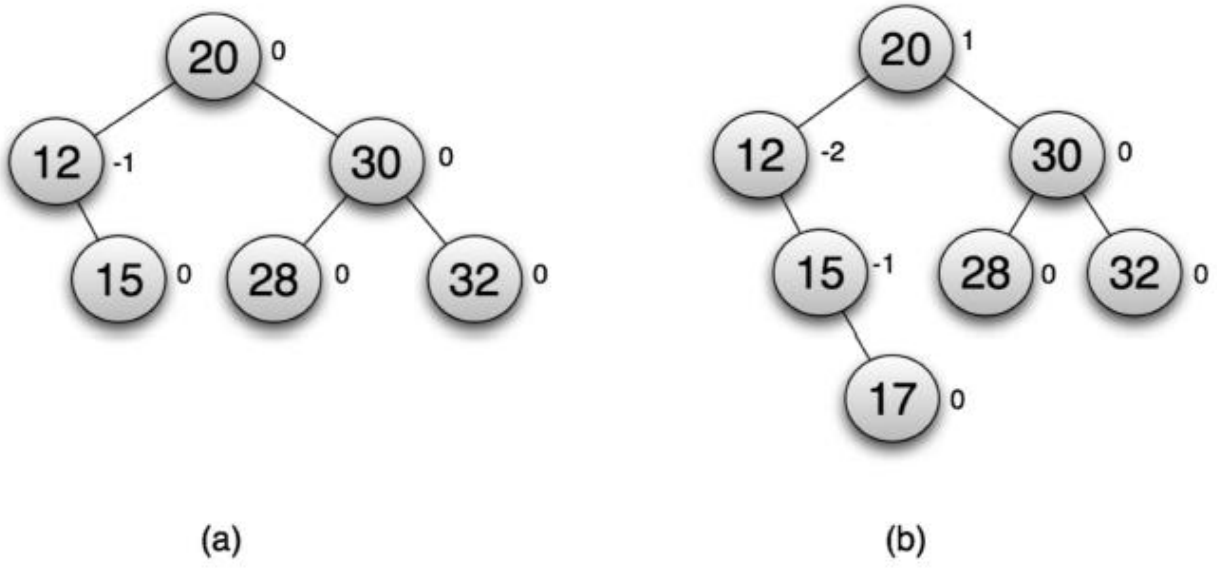
Cálculo do Fator:

- Esta árvore **não é balanceada**: nodo 30, **fator -2**;
- Fator nodo 30: h(esq(0)) – h(dir(2)) = -2.
- Fator nodo 20: h(esq(2)) – h(dir(3)) = -1



Operações devem garantir balanceamento:

- **Árvore (a):** balanceada;
- **Árvore (b):** inserção do **nodo 17**: não balanceada. **Nodo 12** com **fator -2**.



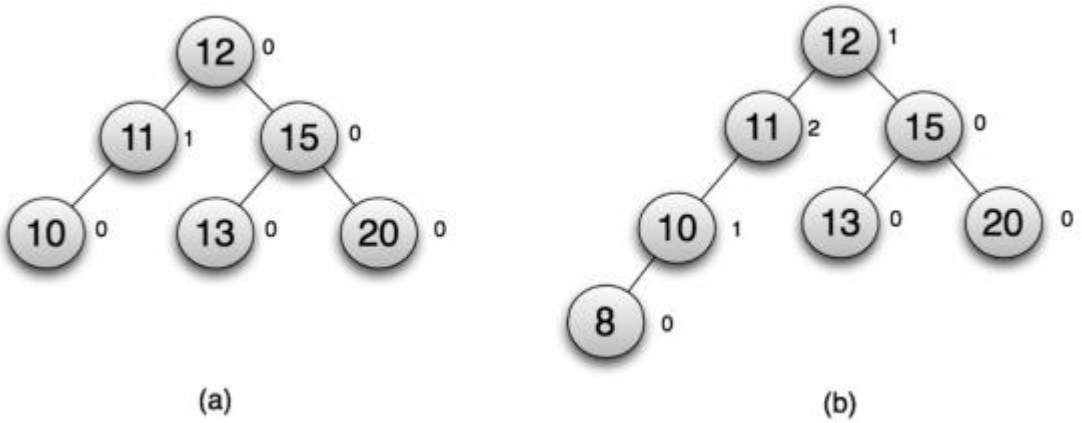
Rotações

- As **rotações garantem o balanceamento** após alguma **operação** na árvore.

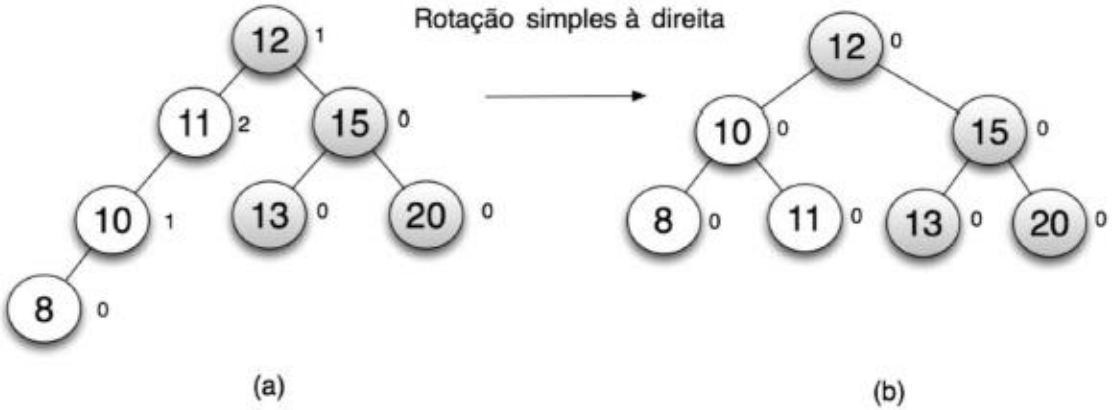
- **Tipos de rotações, para diferentes situações:**
  - Rotação simples à esquerda;
  - Rotação simples à direita;
  - Rotação dupla à esquerda;
  - Rotação dupla à esquerda.

Rotação simples à direita

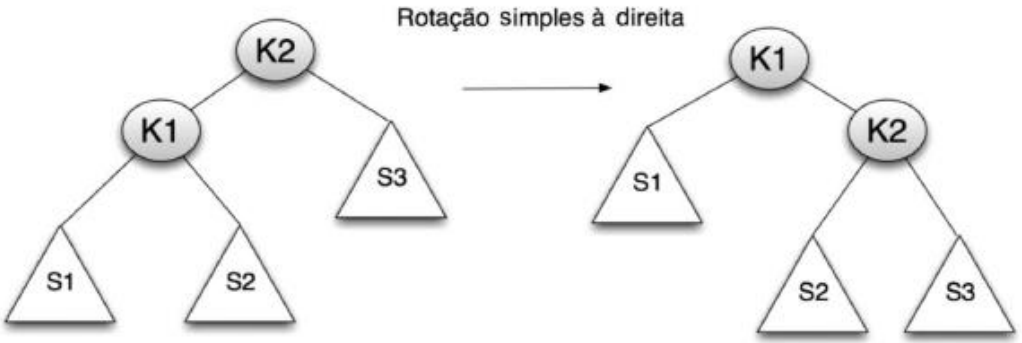
- Uma subárvore com **fator positivo fora** do intervalo.
- Sua **subárvore esquerda** possui **fator positivo**.



Rotação simples à direita

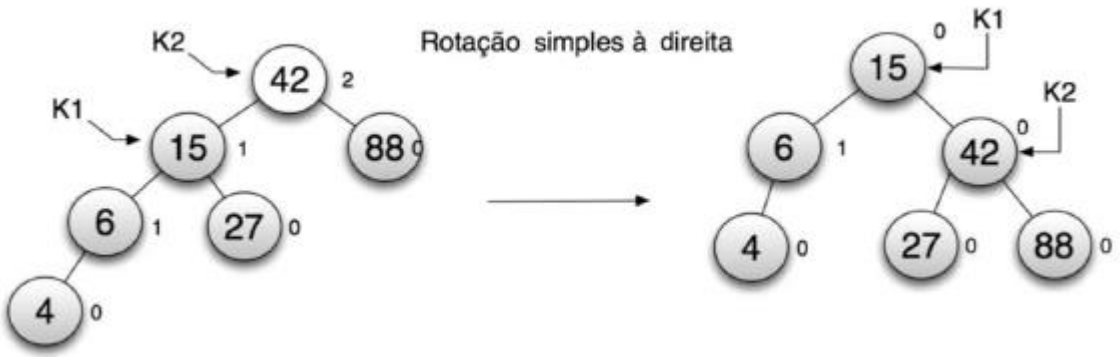


- Para K2, nodo raiz da rotação = 11:
- K1 = filho esquerdo de K2 = 10, torna-se raiz da subárvore
- O filho esquerdo de K2 aponta para o filho direito de K1
- O filho direito de K1 aponta para aK2



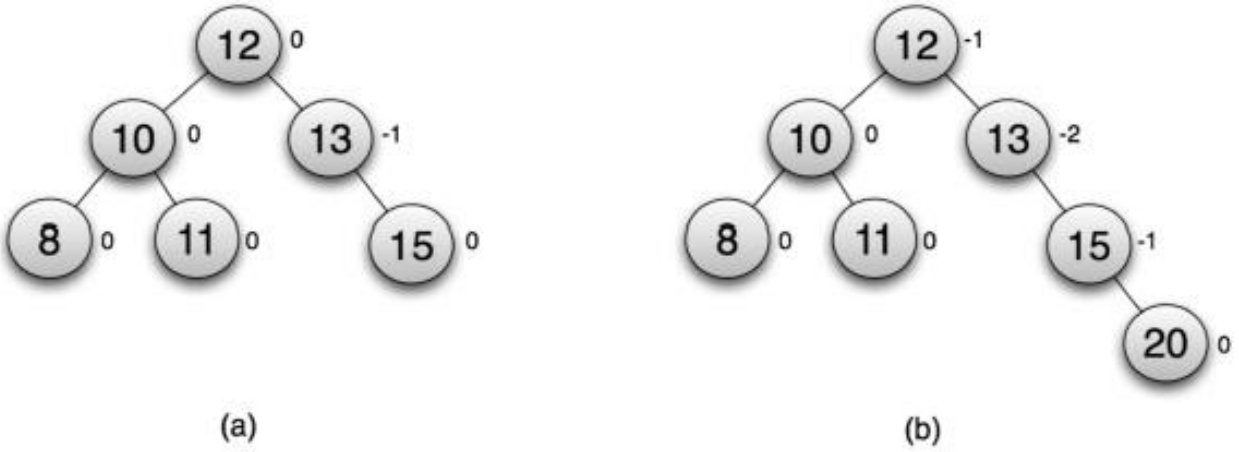
Rotação à direita

```
struct nodo *rotacaodireita(struct nodo *k2){
        struct nodo *k1 = k2->esquerda;
        k2->esquerda = k1->direita;
        k1->direita = k2;
        k2->alt = maximo(altura(k2->esquerda),altura(k2->direita))+1;
        k1->alt = maximo(altura(k1->esquerda), k2->alt)+1;
        return k1;
}
```

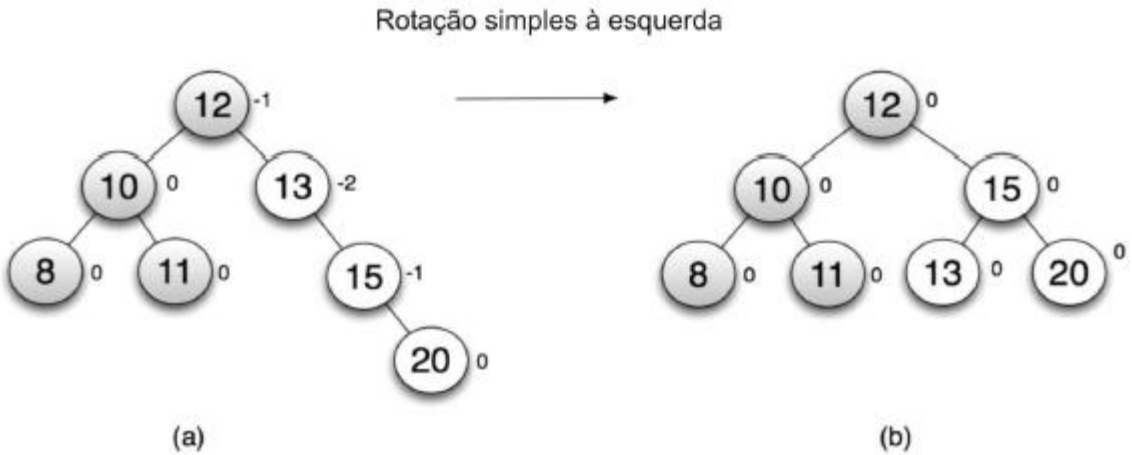


Rotação simples à esquerda

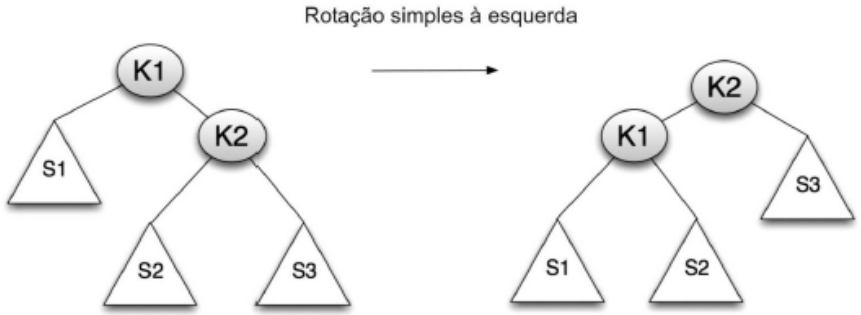
- Uma subárvore com **fator negativo fora** do intervalo.
- Sua **subárvore direita** possui **fator negativo**.



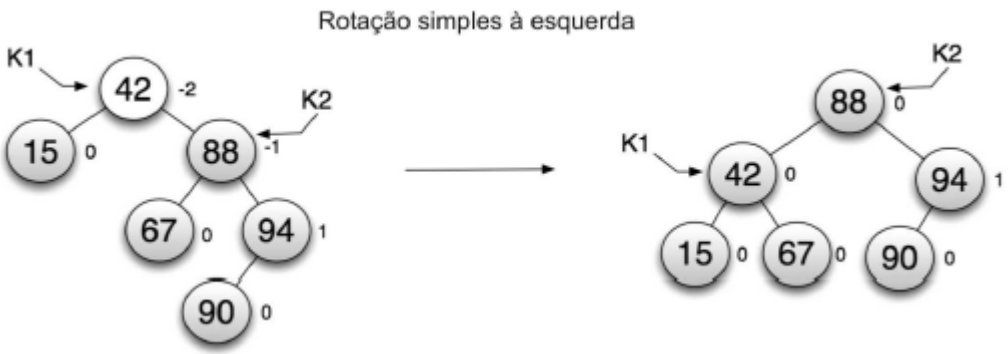
Rotação simples à esquerda



- Para K1, nodo raiz da rotação = 13:
- K2 = filho direito de K1 = 15, torna-se raiz da subárvore
- O filho direito de K1 aponta para o filho esquerdo de K2
- filho esquerdo de K2 aponta para K1



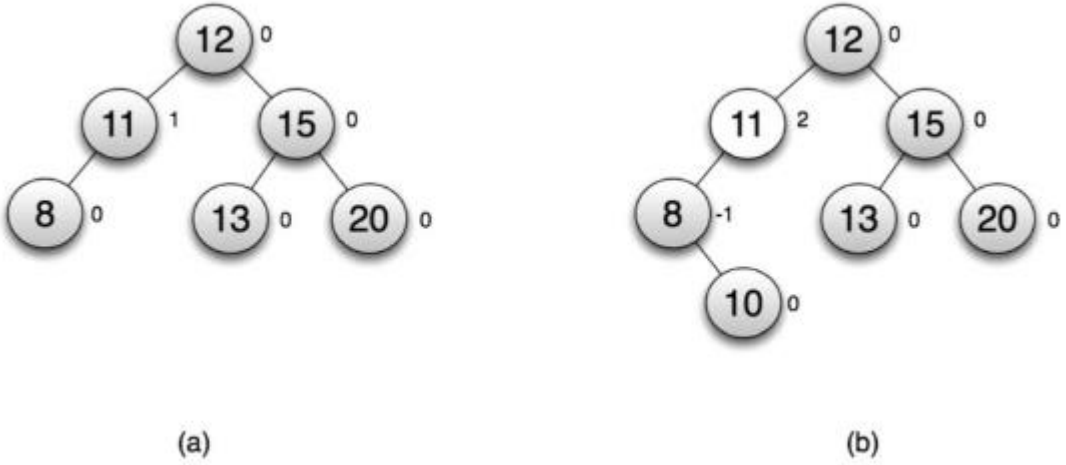
Rotação à esquerda



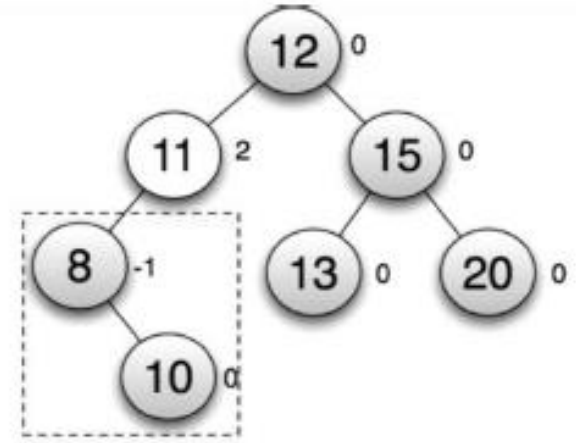

```
struct nodo *rotacaoesquerda(struct nodo *k1){
        struct nodo *k2 = k1->direita;
        k1->direita = k2->esquerda;
        k2->esquerda = k1;
        k1->alt = maximo(altura(k1->esquerda),altura(k1->direita))+1;
        k2->alt = maximo(altura(k2->direita), k1->alt)+1;
        return k2;
}
```

Rotação dupla à direita

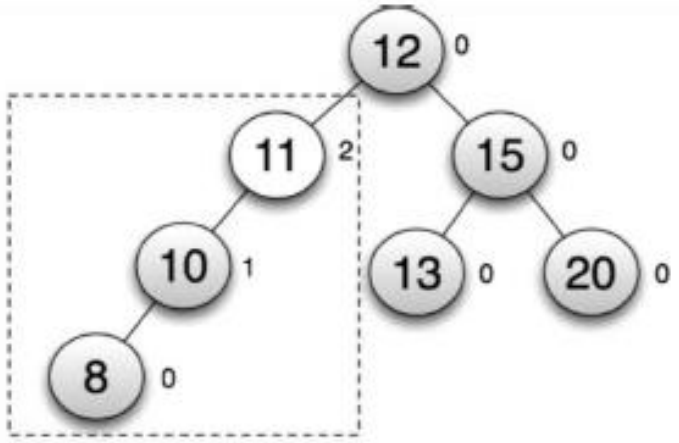
- Uma subárvore com **fator positivo fora** do intervalo.
- Sua **subárvore esquerda** possui **fator negativo**.
- Uma operação dupla equivale a duas simples:
  - Uma simples à esquerda
  - Uma simples à direita



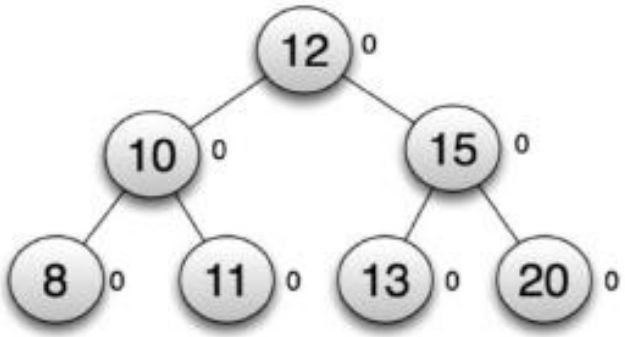
Rotação dupla à direita



(a) Passo 1
Rotação simples à esquerda



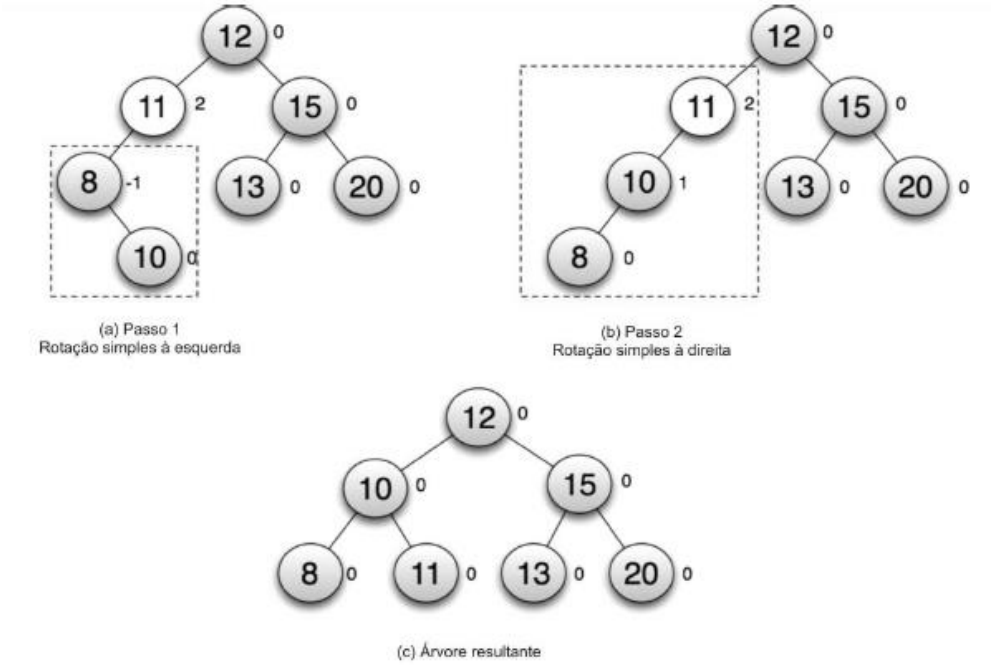
(b) Passo 2
Rotação simples à direita



(c) Árvore resultante

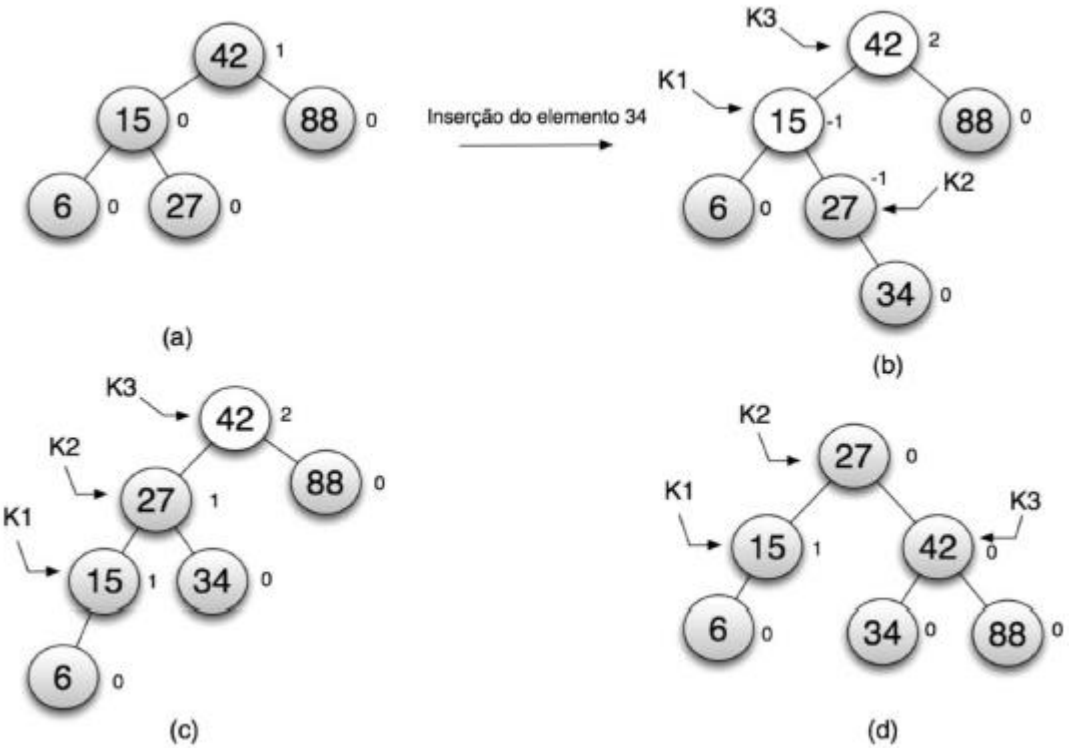
Rotação dupla á direita

- Rotação dupla com raiz da rotação em K3
- Seja K1 filho esquerdo de K3
- Realizar a rotação simples à esquerda em K1.
- Realizar rotação simples à direita em K3.



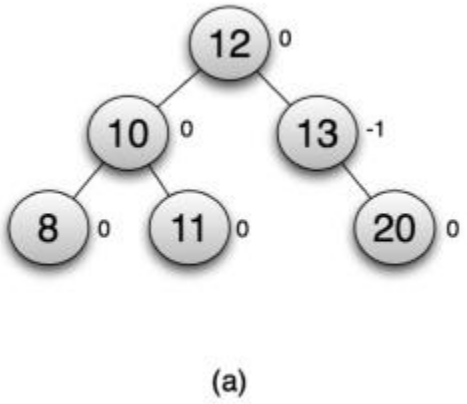
Rotação dupla à direita

```
struct nodo *rotacaodupladireita(struct nodo *k3){
        struct nodo *k1 = k3->esquerda;
        k3->esquerda = rotacaoesquerda(k1);
    return rotacaodireita(k3);
}
```

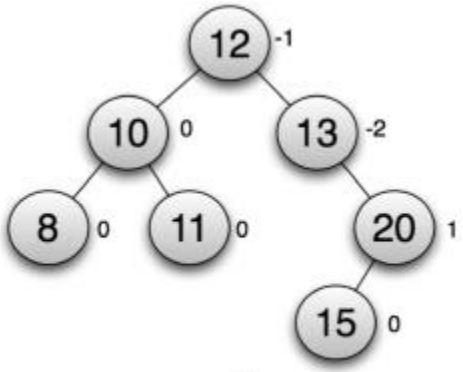


Rotação dupla à esquerda

- Uma subárvore com **fator negativo fora** do intervalo.
- Sua **subárvore direita** possui **fator positivo**
- Uma operação dupla equivale a duas simples:
  - Uma simples à direita
  - Uma simples à esquerda

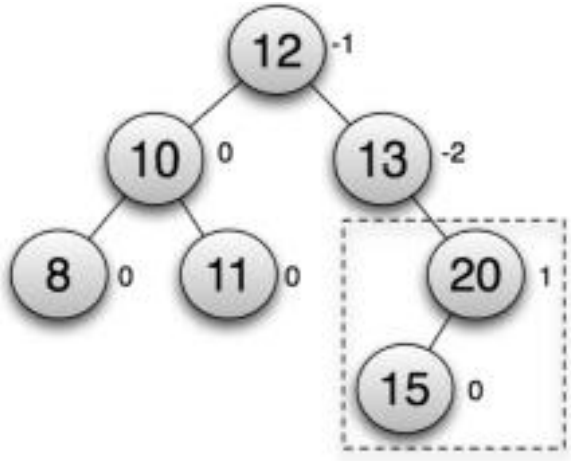


(a)

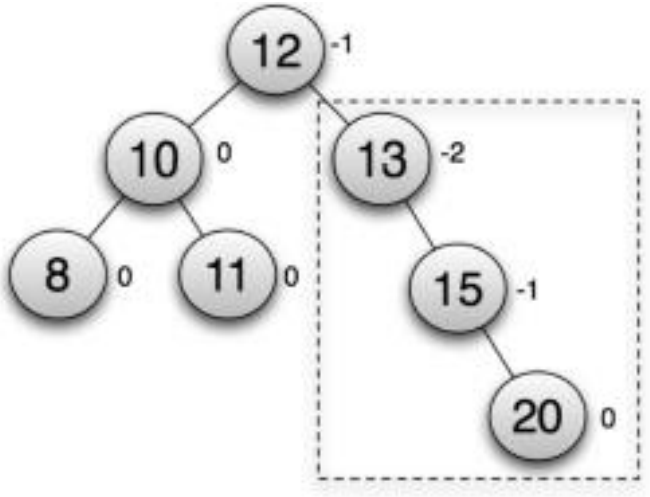


(b)

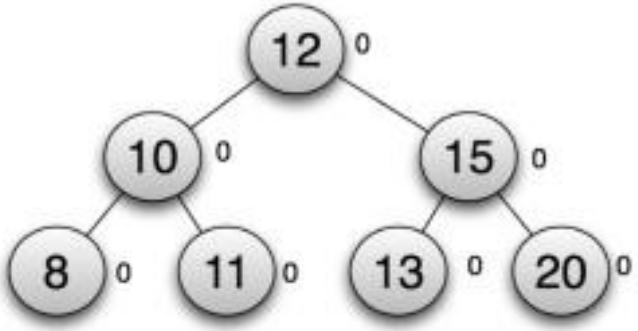
Rotação dupla à esquerda



(a) Passo 1
Rotação Simples à Direita



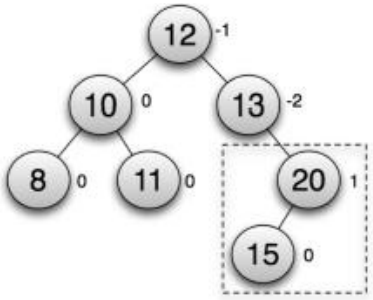
(b) Passo 2
Rotação Simples à Esquerda



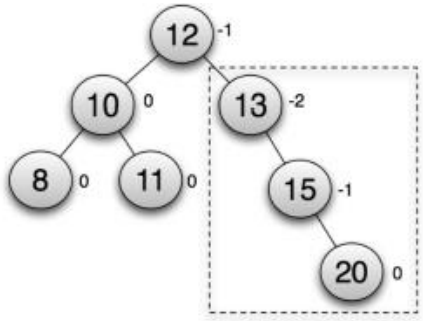
(c) Árvore Resultante

Rotação dupla à esquerda

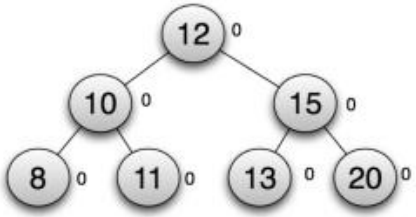
- Rotação dupla com raiz da rotação em K1
- Seja K3 filho direito de K1
- Realizar a rotação simples à direita em K3.
- Realizar rotação simples à esquerda em K1.



(a) Passo 1
Rotação Simples à Direita



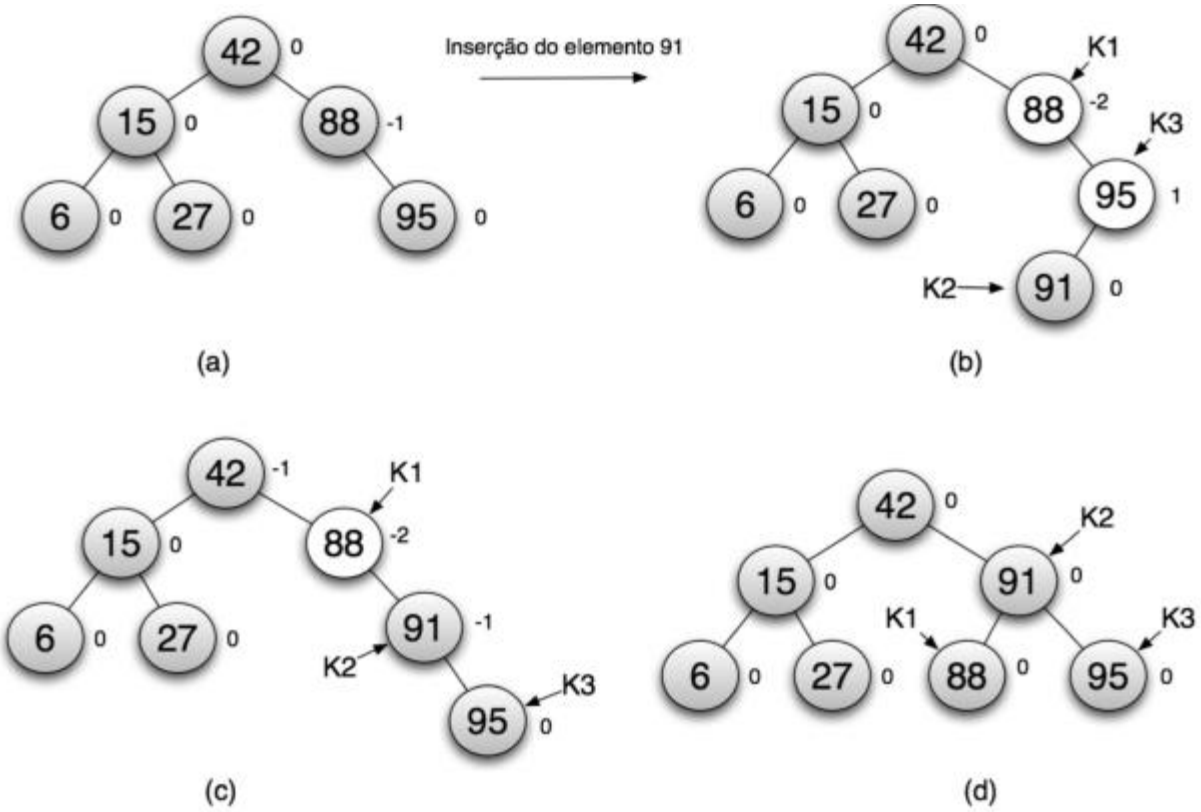
(b) Passo 2
Rotação Simples à Esquerda



(c) Árvore Resultante

Rotação dupla à esquerda

```
struct nodo *rotacaoduplaesquerda(struct nodo *k1){
        struct nodo *k3 = k1->direita;
        k1->direita = rotacaodireita(k3);
    return rotacaoesquerda(k1);
}
```



Inserção de elemento

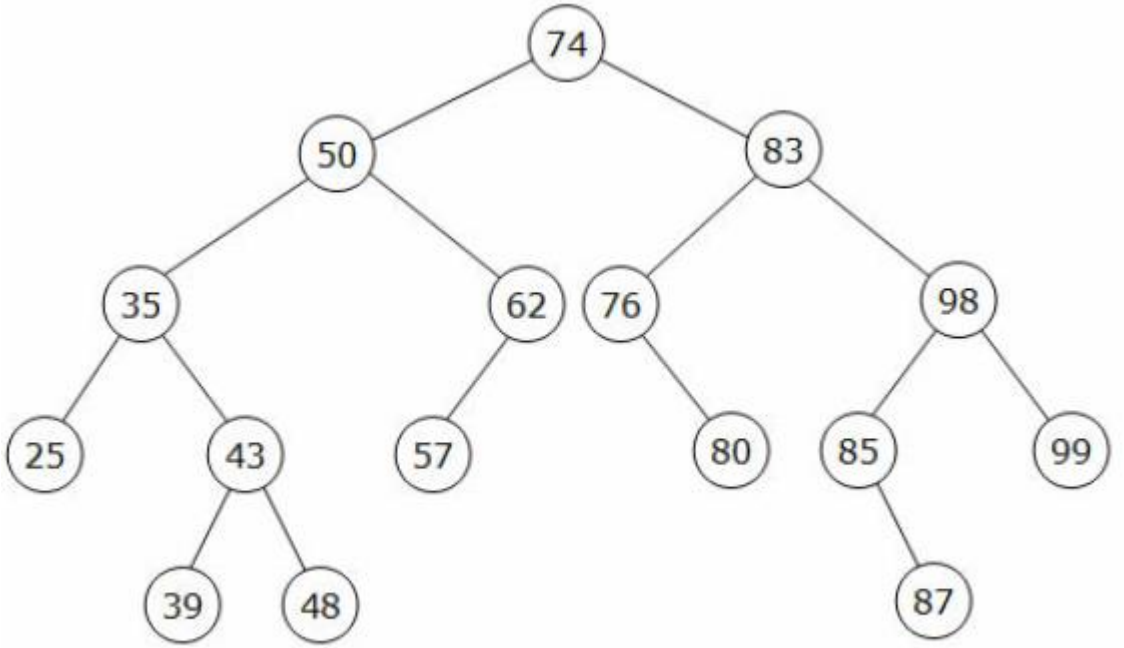
- Primeiro insere o elemento.
- Depois verifica se é necessário aplicar rotação.

```
struct nodo *insere (struct nodo *raiz, int el){
        struct nodo *p=raiz;
        if (raiz == NULL) p=crianodo(el);
        else if(el<p->chave)  p->esquerda = insere(p->esquerda,el);
        else if(el>p->chave)  p->direita = insere(p->direita, el);

        if(getfator(p)==2){  //fator externo fora do limite e positivo
                if(getfator(p->esquerda)>0) p=rotacaodireita(p);
                else p= rotacaodupladireita(p);
        } else
        if(getfator(p)==-2){  //fator externo fora do limite e negativo
                if(getfator(p->direita)<0) p=rotacaoesquerda(p);
                else p= rotacaoduplaesquerda(p);
        }
        p->alt = maximo(altura(p->esquerda), altura(p->direita))+1;
        return p;
}
```

Exercício

- Dada a árvore **AVL** abaixo, insira o nó **37**, garantindo que a árvore resultante também **será uma árvore AVL.** Indique (se necessário) a operação de **rotação** realizada, **justificando** sua utilização.



Exercício

- Monte (desenhe) passo a passo a árvore AVL resultante apos a inserção das seguintes chaves na ordem em que elas aparecem. Em cada passo indique, caso seja necessário, qual foi a rotação que ocorreu.
- **10, 15, 13, 7, 6, 18, 17**