INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO PIAUÍ

CAMPUS TERESINA-CENTRAL

DIRETORIA DE ENSINO

Estrutura de Dados II – **Balanceamento com o Algoritmo DSW**

- Aula 10 -

Professora: Elanne Cristina O. dos Santos

elannecristina.santos@gmail.com
elannecristina.santos@ifpi.edu.br

Algoritmo de Balanceamento

- Existem inúmeros algoritmos para casos diferentes de situações. A forma básica é mapear a árvore para um vetor e reconstruir a partir de suas subdivisões.

- O algoritmo de balanceamento visto anteriormente tem a desvantagem de exigir **UMA MATRIZ ADICIONAL** que necessitava **SER ORDENADA ANTES QUE A CONSTRUÇÃO DE UMA ÁRVORE PERFEITAMENTE BALANCEADA** começasse.

- Pode ser construída a matriz inicialmente, mas nesse caso pode ser impróprio quando a árvore tem que ser usada enquanto os dados a serem incluídos nela ainda estão chegando.

Algoritmo de Balanceamento

- Uma solução seria: transferir os dados da árvore desbalanceada direto para a matriz usando o percurso in-order (LVR), isso evitaria da necessidade do algoritmo de ordenação.Depois remover a árvore e reconstruí-la usando o algoritmo de balanceamento.

- Para evitar a ordenação, portanto, ele exigia a demolição e então reconstrução da árvore, o que é ineficiente, exceto para árvores bem pequenas.

Algoritmo DSW

- O algoritmo **DSW** (criadores do algoritmo: Day, Sout e Warren) exige PEQUENA ARMAZENAGEM ADICIONAL PARA VARIÁVEIS INTERMEDIÁRIAS E USO DE PROCEDIMENTOS SEM ORDENAÇÃO.

- O essencial neste algoritmo é a **rotação**, que pode ser à esquerda ou à direita do nó com relação ao seu ascendente.

Algoritmo da rotação à direita:

rotateRight(Gr, Par, Ch) {

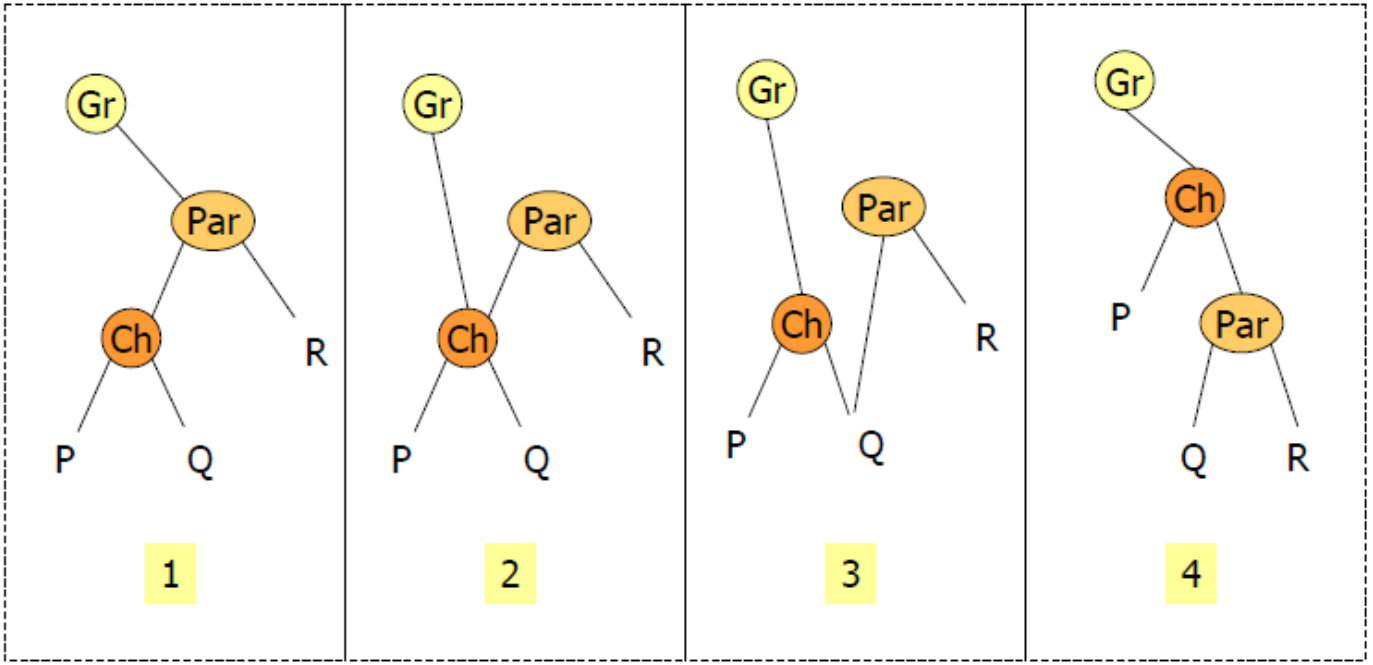
if Par não é raiz

o avô Gr do filho Ch se torna o ascendente de Ch substituindo Par;

a subárvore direita de Ch se torna a árvore esquerda do ascendente Par de Ch;

o nó Ch obtém Par como seu filho direito;

}



Rotação a Direita

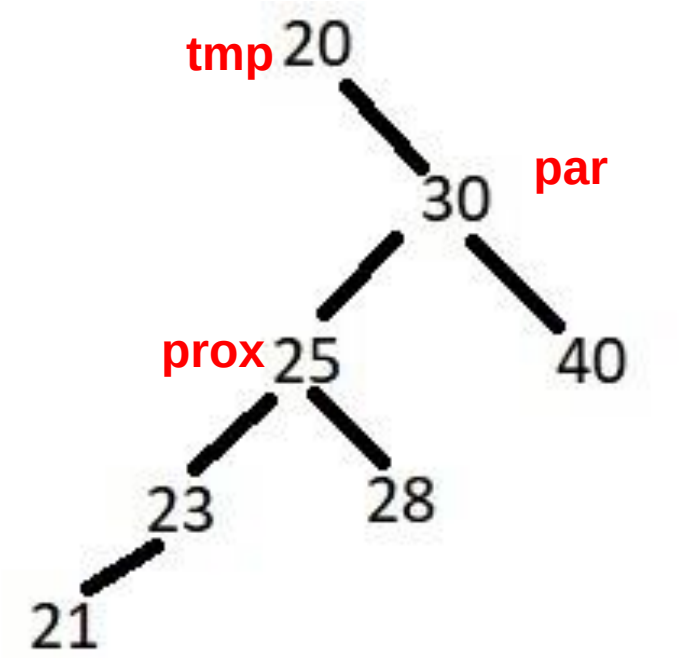
ArvoreNo<T> *tmp=p;

ArvoreNo<T> *par;

ArvoreNo<T> *prox;

ArvoreNo<T> *subD;

par = tmp->right;



Rotação a Direita

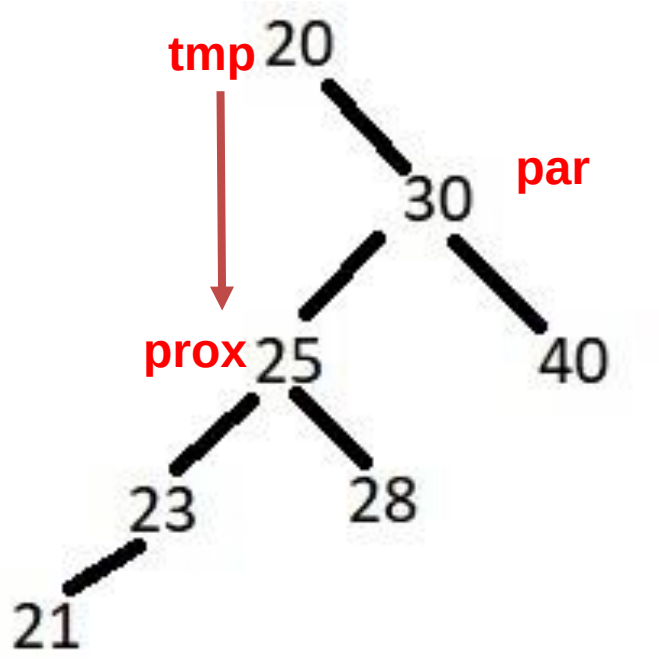
ArvoreNo<T> *tmp=p;

ArvoreNo<T> *par;

ArvoreNo<T> *prox;

ArvoreNo<T> *subD;

par = tmp->right;



-> tmp->right = prox

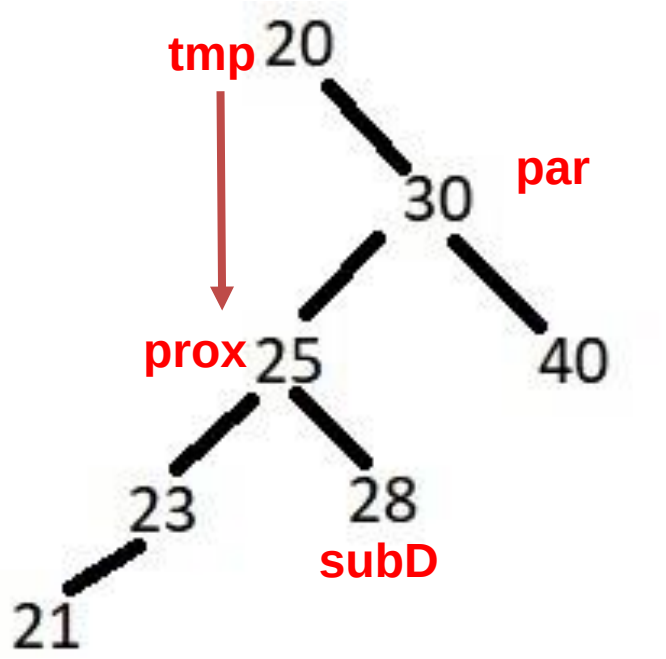
subD=prox->right;

par->left =subD;

prox->right = par;

Rotação a Direita

```
ArvoreNo<T> *tmp=p;
ArvoreNo<T> *par;
ArvoreNo<T> *prox;
ArvoreNo<T> *subD;
par = tmp->right;
```




```
tmp->right = prox;
-> subD=prox->right;
par->left =subD;
prox->right = par;
```

Rotação a Direita

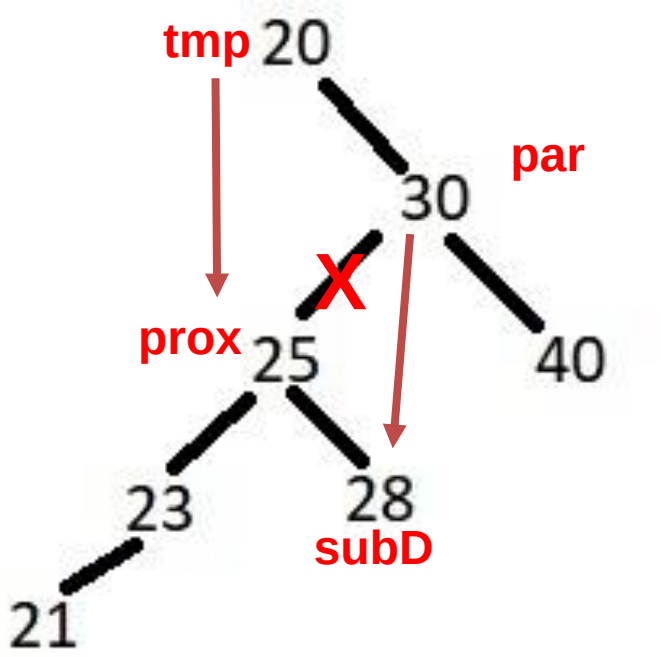
ArvoreNo<T> *tmp=p;

ArvoreNo<T> *par;

ArvoreNo<T> *prox;

ArvoreNo<T> *subD;

par = tmp->right;



tmp->right = prox;

subD=prox->right;

-> par->left =subD;

prox->right = par;

Rotação a Direita

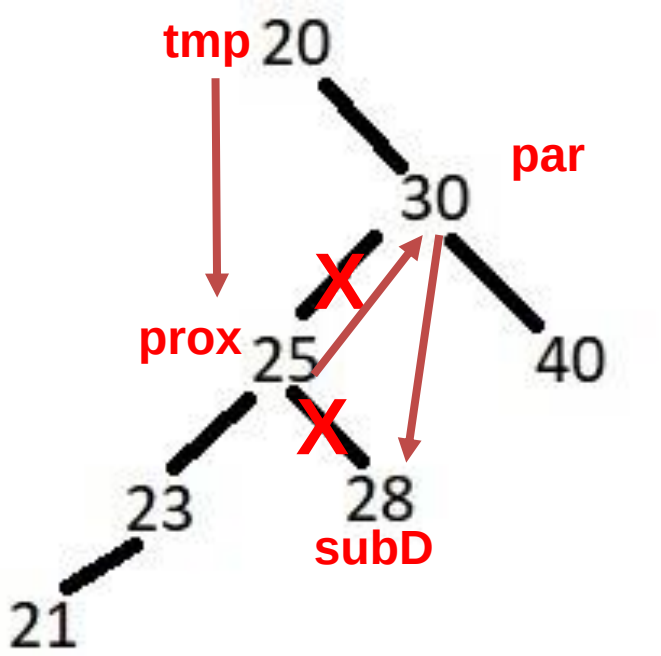
ArvoreNo<T> *tmp=p;

ArvoreNo<T> *par;

ArvoreNo<T> *prox;

ArvoreNo<T> *subD;

par = tmp->right;



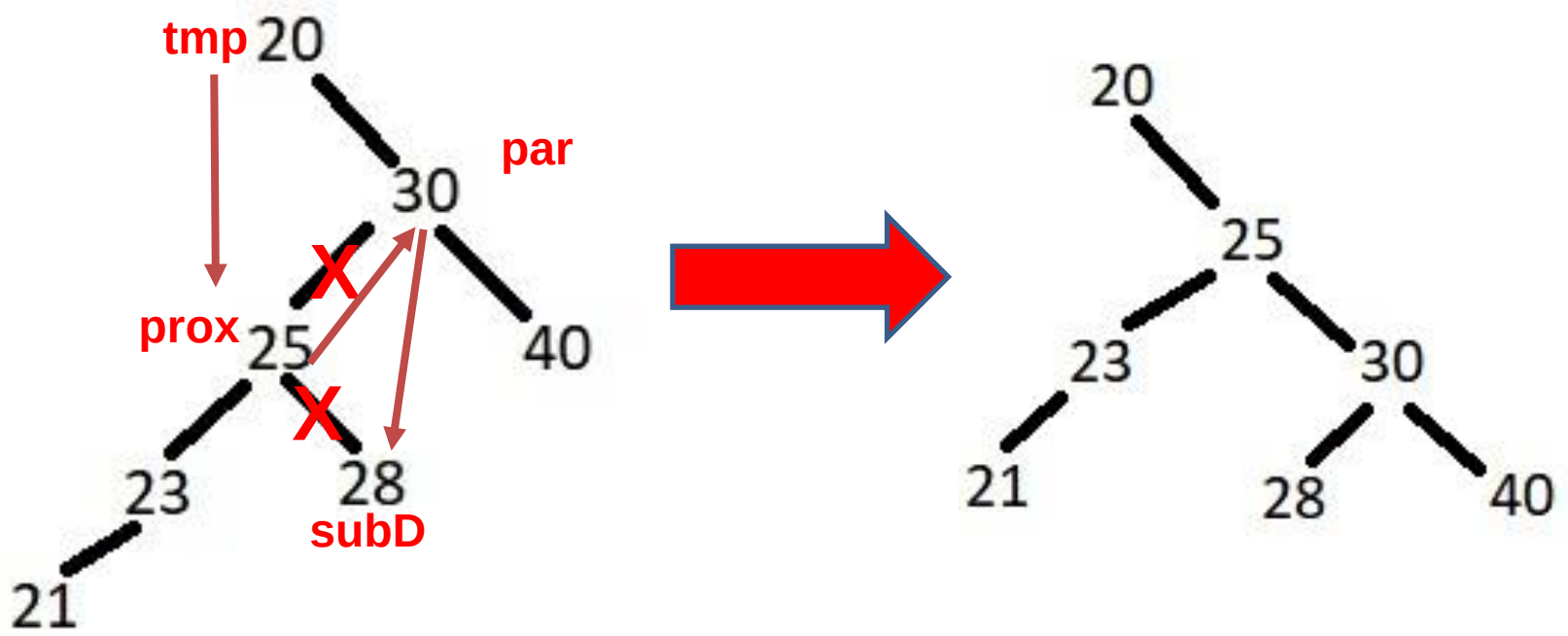
tmp->right = prox;

subD=prox->right;

par->left =subD;

-> prox->right = par;

Rotação a Direita



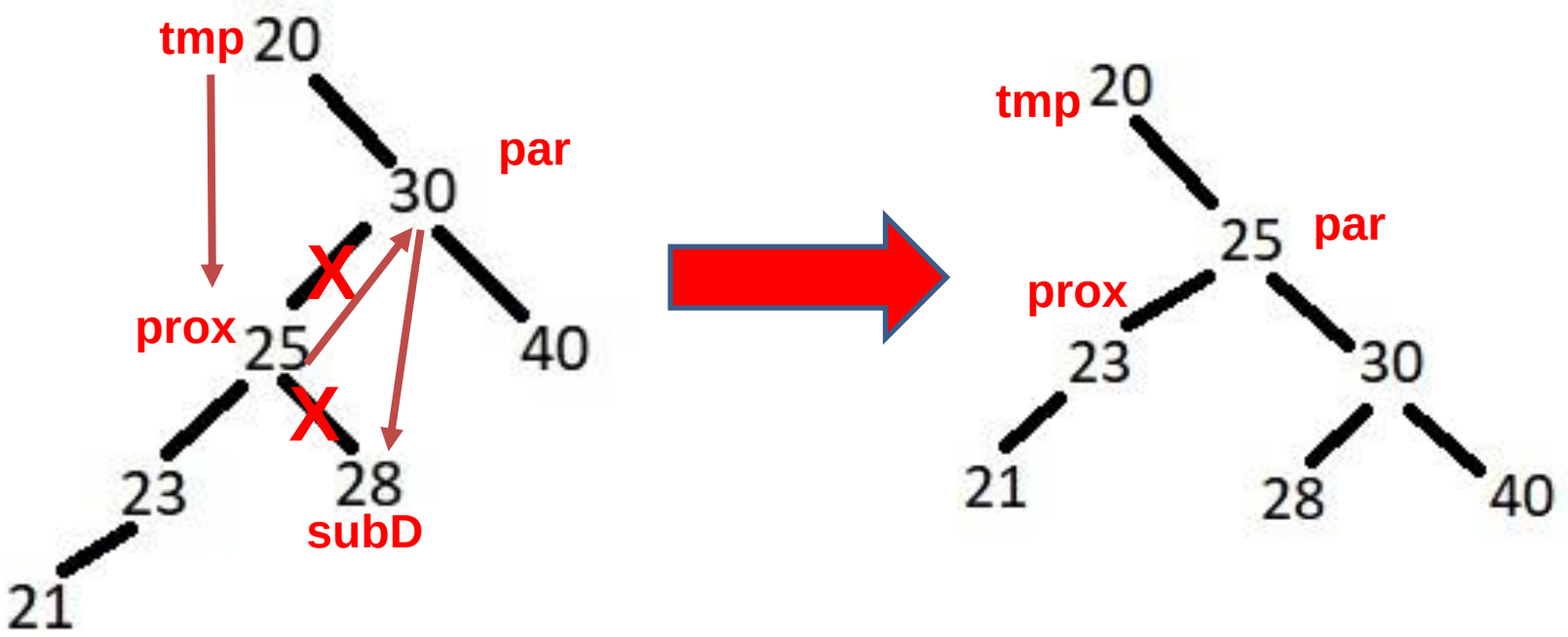
tmp->right = prox;

subD=prox->right;

par->left =subD;

prox->right = par;

Rotação a Direita



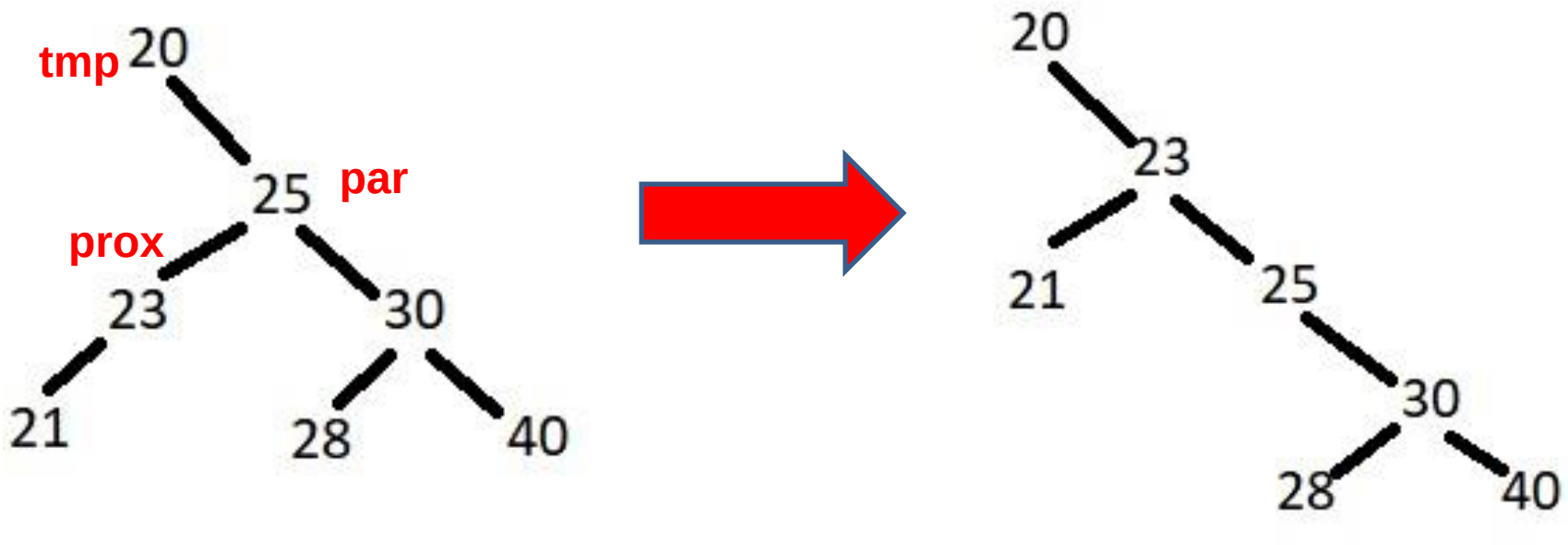
tmp->right = prox;

subD=prox->right;

par->left =subD;

prox->right = par;

Rotação a Direita

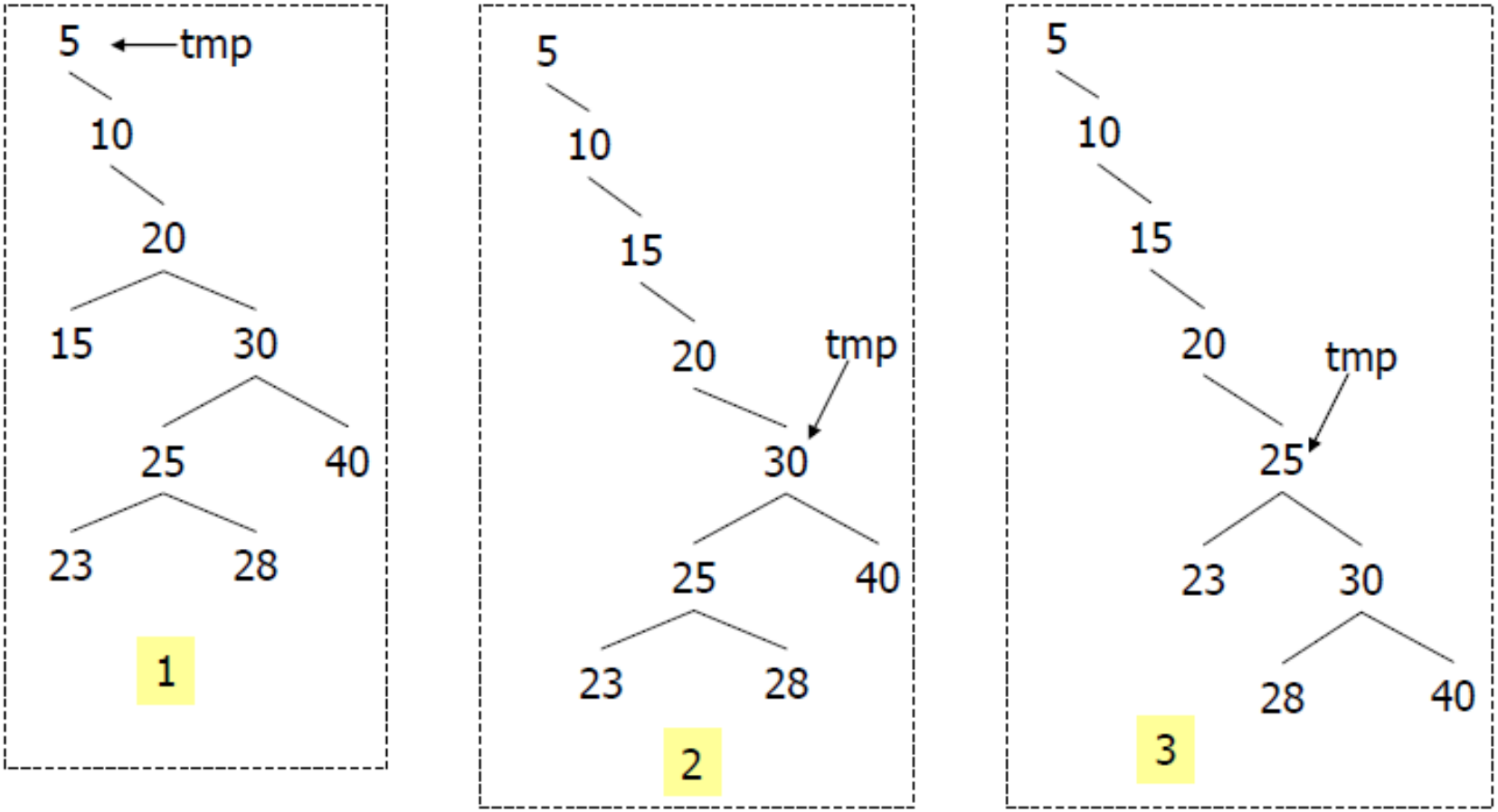


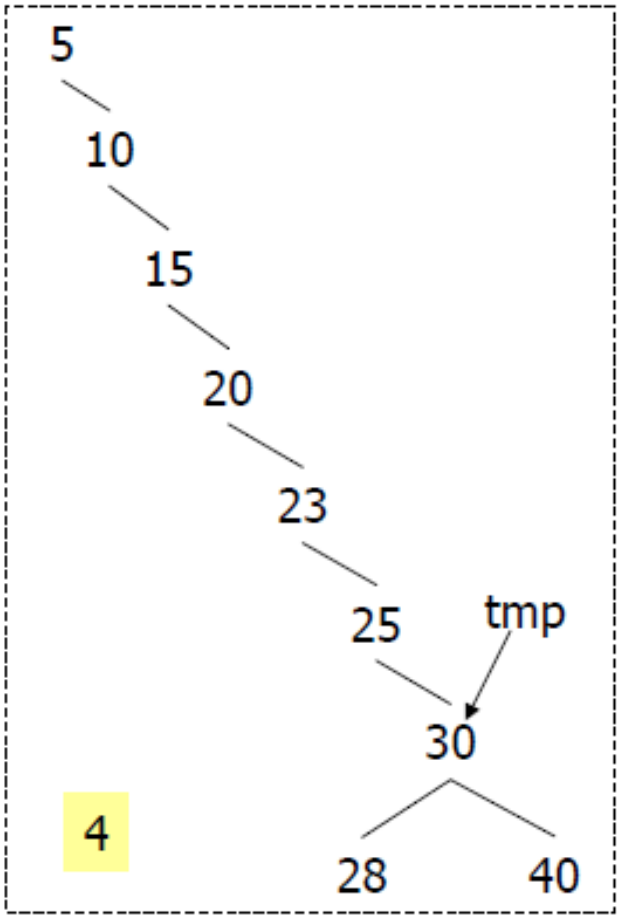

```
tmp->right = prox;
subD=prox->right;
par->left =subD;
prox->right = par;
```

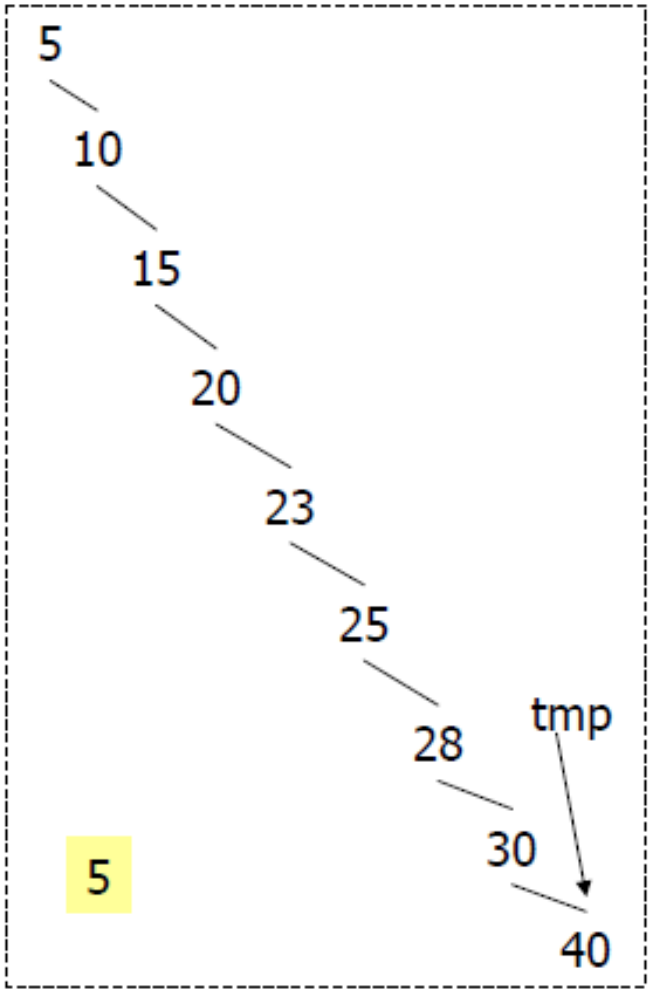
Algoritmo DSW

- Na 1ª fase ele resulta em um árvore parecida com uma lista ligada chamada de *espinha dorsal*.

```
createBackbone (root, n) {
  tmp = root;
  while (tmp != 0) {
    if tmp tem um filho esquerdo
      gire esse filho ao redor de tmp;
      ajuste tmp para o filho que se tornou
      ascendente;
    else ajuste tmp para o filho direito;
  }
}
```


5
tmp
10
20
15
30
25
40
23
28
1
5
10
15
20
tmp
30
25
40
23
28
2
5
10
15
20
tmp
25
23
30
28
40
3


5
10
15
20
23
25
tmp
30
4
28
40


5
10
15
20
23
25
28
tmp
30
5
40

Algoritmo DSW

- Na 2ª fase a espinha dorsal é transformada em uma árvore balanceada:

```
createPerfectTree (root, n) {
  m = (2 elevado ao menor valor de lg(n+1)) - 1;
  faça n - m rotações começando da raiz;
  while (m > 1) {
    m = m / 2;
    faça m rotações começando da raiz;
  }
}
```

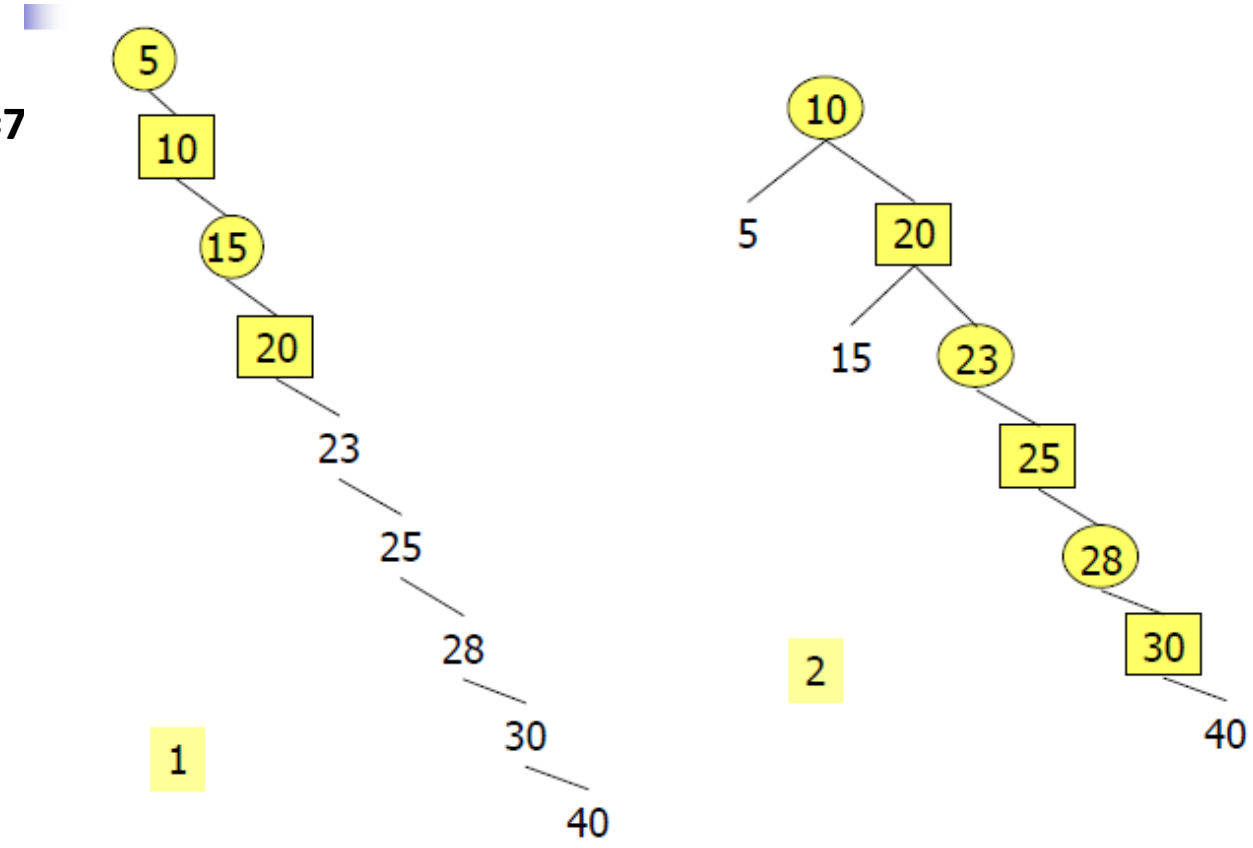
Algoritmo DSW

```
m = (2 elevado ao menor valor de lg(n+1)) - 1;
faça n - m rotações começando da raiz;
```

n=9

m =(2 elevado a *log* 10) - 1 =7

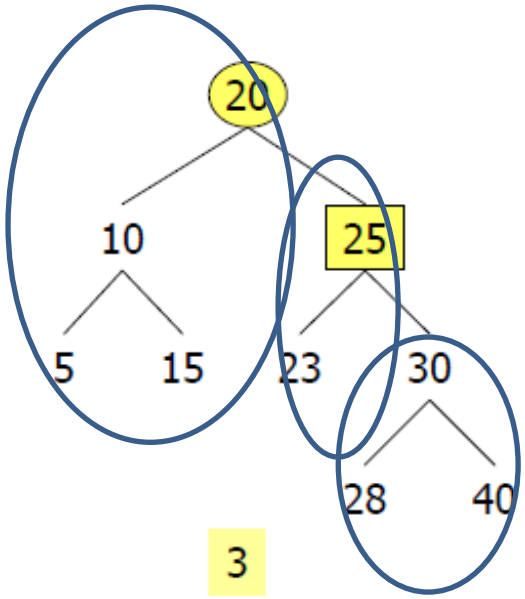
Rotações =9-7=2



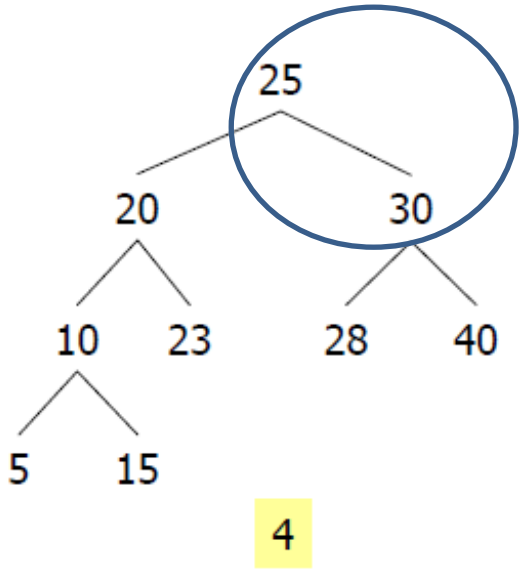
Algoritmo DSW

```
while (m > 1) {
  m = m / 2;
  faça m rotações começando da raiz;
}
```

m=7/2 =>3



m=3/2 =>1



Algoritmo DSW

- O número de rotações pode ser calculado pela formula n - |log(n+1)|, ou seja, o número de rotações é O(n).
- Criar a espinha dorsal exige no máximo O(n) rotações, por isso o custo do rebalanceamento global é ótimo em termos de tempo, cresce linearmente com n e exige pequena e fixa quantidade de armazenagem.

Atividade

1) Inclua os seguintes valores na seguinte ordem em uma árvore binária:

8, 20, 25, 14, 32, 30,28,31,40.

1.1) Qual a altura da árvore resultante?

1.2) Mostre a árvore resultante.

1.3) A árvore resultante está balanceada ou não?

1.4) Aplique o algoritmo DSW se a arvore estiver desbalanceada. Qual a árvore resultante?