```
void                              precursoEmLarguraColocacaoEmVetor
 (ARV_BIN_BUSCA arvore, int vetor[], int *num_elem)
{
   FILA_ENC fila;
   cria_fila(&fila);
   if (arvore)
   {
      ins_fila (fila, arvore);
      *num_elem = 0;
      vetor = NULL;
   }
   while (!eh_vazia_fila(fila))
   {
      vetor = (int *) realloc (vetor, (++(*num_elem))* sizeof(int));
      vetor[(*num_elem)-1] = info(cons_fila(fila));
      if (left(cons_fila(fila)))
         ins_fila (fila, left(cons_fila(fila)));
      if (right(cons_fila(fila)))
         ins_fila (fila, right(cons_fila (fila)));
      ret_fila(fila);
   }
}
```

```
int particionar (int v[], int ii, int is) {
   int esq=ii, dir=is, pivo=v[ii];
   while (esq<dir) {
      while (v[esq]<=pivo && esq<is)
         esq++;
      while (v[dir]>pivo)
         dir--;
      if (esq<dir) {
         int temp;
         temp = v[esq];
         v[esq]=v[dir];
         v[dir]=temp;
      }
   }
   v[ii]=v[dir];
   v[dir]=pivo;
   return dir;
}
```

```
void quicksort (int *v, int n)
{
  if (n>1)
  {
    int pont_part=particionar(v, 0, n-1);
    quicksort (v, pont_part);
    quicksort (&v[pont_part+1],
      n-1-pont_part);
  }
}
```

```
void balancearArvoreBinariaAux (ARV_BIN_BUSCA *arvore,
int vetor[], int inicio, int fim)
{
  if (inicio <= fim)
  {
        int meio = (inicio+fim)/2;
        ins_ele(arvore, vetor[meio]);
        balancearArvoreBinariaAux (arvore, vetor, inicio,
        meio-1);
        balancearArvoreBinariaAux (arvore, vetor, meio+1,
        fim);
  }
}
```

```
void balancearArvoreBinaria (ARV_BIN_BUSCA *arvore)
{
   int *vetor, num_elem;
   precursoEmLarguraColocacaoEmVetor (*arvore, vetor,
   &num_elem);
   quicksort (vetor, num_elem);
   while (*arvore)
      remocaoPorCopia (arvore);
   balancearArvoreBinariaAux (arvore, vetor, 0, num_elem-1);
   free(vetor);
}
```

Árvore Balanceada

O algoritmo apresentado possui um sério inconveniente, pois, caso a árvore já exista, os seus elementos devem ser retirados e colocado em um vetor, para que a mesma seja recriada.

Caso a árvore ainda não exista, o inconveniente ainda persiste, pois todos os dados precisam ser colocados em um vetor antes da árvore ser criada.

Uma pequena melhoria pode ser feita, pois mesmo que a árvore binária de busca esteja desbalanceada, se for efetuado um percurso in-ordem elimina-se a necessidade de ordenar o vetor.

UNIVASF

Árvore Balanceada

Existem formas mais eficientes de se balancear uma árvore.

Um exemplo é o algoritmo desenvolvido por Colin Day e posteriormente melhorado por Quentin F. Stout e Bette L. Warren.

Denominado Algoritmo DSW, devido aos nomes de seus idealizadores, baseia-se em percorrer uma árvore binária de busca tornando-a uma arvore degenerada (similar a uma lista encadeada) e posteriormente percorrê-la novamente tornando-a uma árvore perfeitamente balanceada.

UNIVASF

Árvore AVL

A seguir estudaremos árvore AVL e árvore 234.

Os slides que versão sobre as árvores retro aludidas foram baseados nos slides gerados pela professora Elisa Maria Pivetta Cantarelli intitulados Árvores Binárias Balanceadas.

Até o momento vimos algoritmos que balanceiam árvores globalmente.

Contudo, o rebalanceamento pode ocorrer localmente se alguma porção da árvore for desbalanceada por uma operação de inserção ou remoção de um elemento da árvore.

Um método clássico para tal foi proposto por Adel'son-Vel'skii e Landis e o nome dado a uma árvore modificada com este método é árvore AVL.

Árvore AVL

Uma árvore AVL é uma árvore binária de busca onde a diferença em altura entre as subárvores esquerda e direita de cada nó é no máximo um (positivo ou negativo).

Esta diferença é chamada de fator de balanceamento (FB).

O FB (ou informações que permitam sua obtenção) é acrescentado a cada nó da árvore AVL.

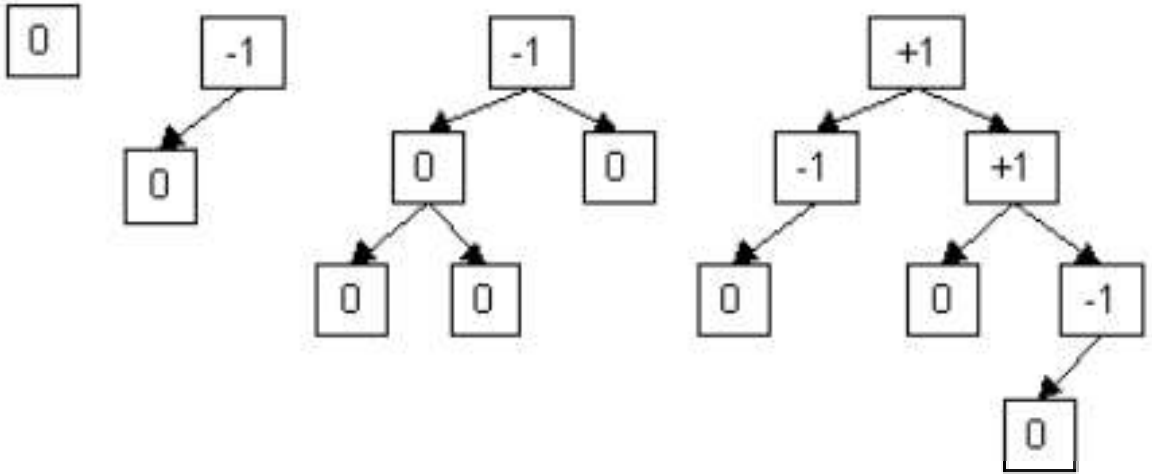
O FB é obtido através da seguinte equação:

FB (nó *p*) = altura(subárvore direita de *p*) - altura(subárvore esquerda de *p*)

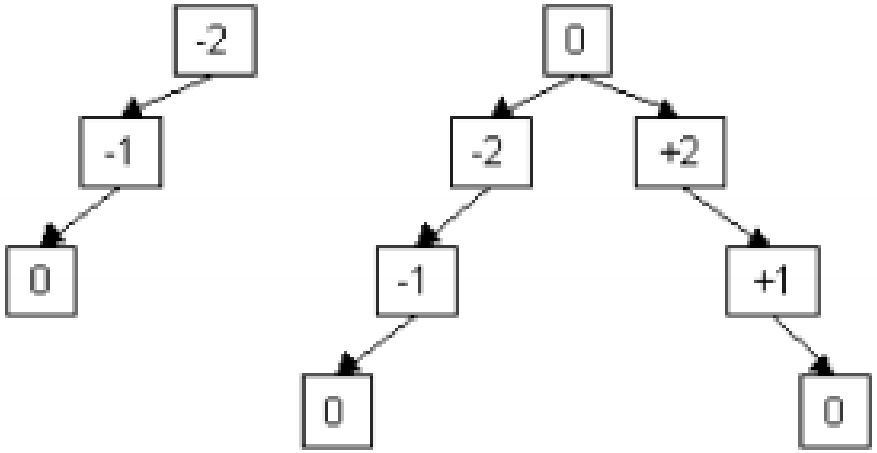
Veremos alguns exemplos de árvores AVL e árvores não AVL.

Árvore AVL

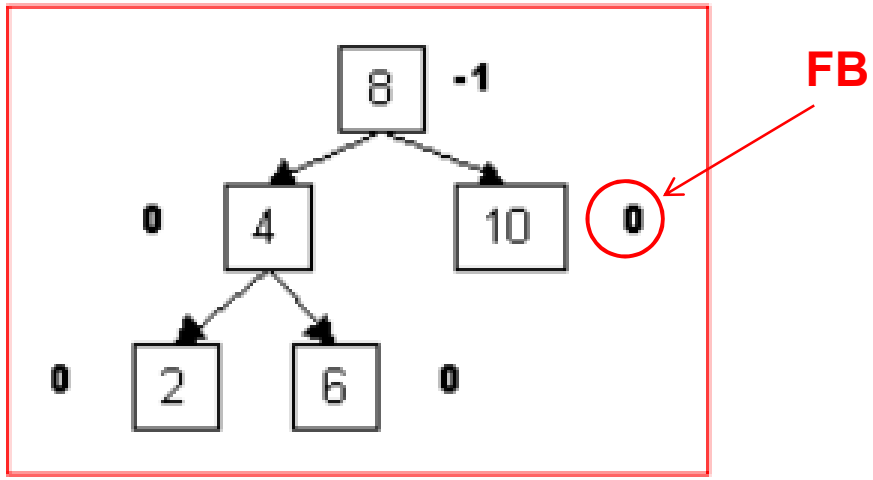
Árvores AVL:



Árvores não AVL:



Árvore AVL



Árvore AVL

Ao inserirmos um novo nó em uma árvore AVL, podemos ou não violar a propriedade de balanceamento.

Caso, ocorra uma violação devemos rebalancear a árvore através da execução de operações de rotação sobre nós da árvore.

Árvore AVL

Após uma inserção que gere um desbalanceamento na árvore AVL podemos nos deparar com duas classes de desbalanceamento.

Onde estas são identificadas com base na análise dos FB's.

Ao inserirmos um novo nó devemos ajustar os FB's, desde o nó inserido até a raiz ou até encontrarmos um fator de balanceamento inaceitável, ou seja, com valor 2 ou -2.

Quando o FB do nó filho com valor 1 (+ ou -) possuir o mesmo sinal do FB de seu pai (nó com FB 2 ou -2) trata-se da classe de desbalanceamento 1, que requer apenas uma rotação simples para a árvore ser rebalanceada.

Árvore AVL

Quando o FB do nó filho com valor 1 (+ ou -) possuir sinal oposto ao FB de seu pai (nó com FB 2 ou -2) trata-se da classe de desbalanceamento 2 que requer uma rotação dupla, ou em outras palavras, duas rotações para a árvore ser rebalanceada.

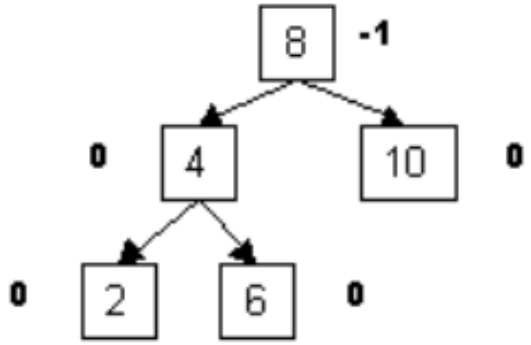
Se o sinal do FB do nó que caracteriza o desbalanceamento for positivo a rotação será para a esquerda.

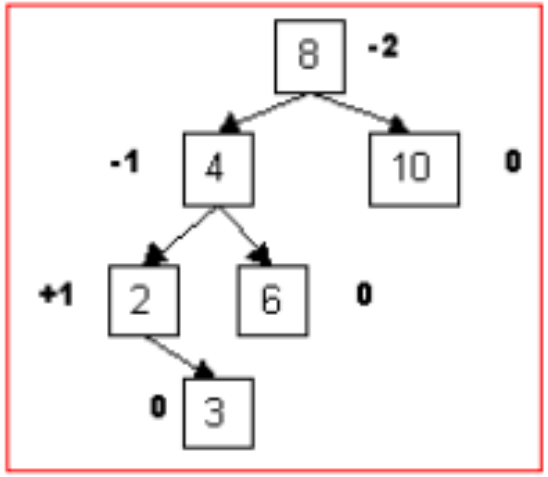
Se o sinal do FB do nó que caracteriza o desbalanceamento for negativo a rotação será para a direita.

Analisaremos agora um exemplo de cada uma das classes citadas.

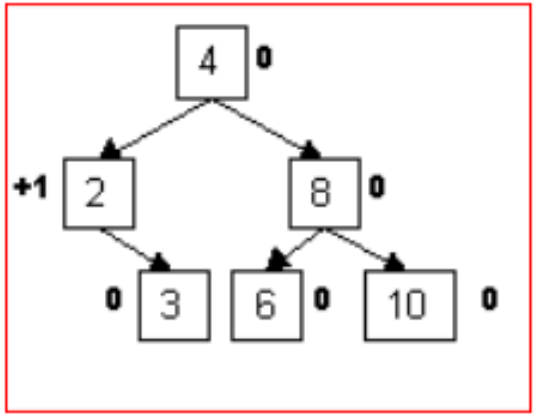
Árvore AVL

Ao inserirmos o valor 3 na árvore abaixo teremos:



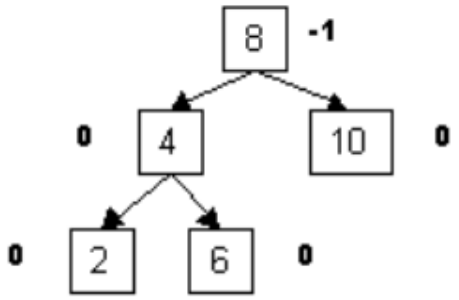


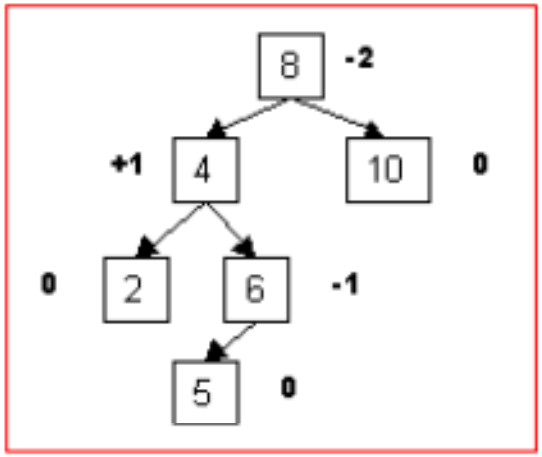
Identificamos a classe 1, que demanda uma rotação simples à direita. Após a rotação teremos:



Árvore AVL

Ao inserirmos o valor 5 na árvore abaixo teremos:



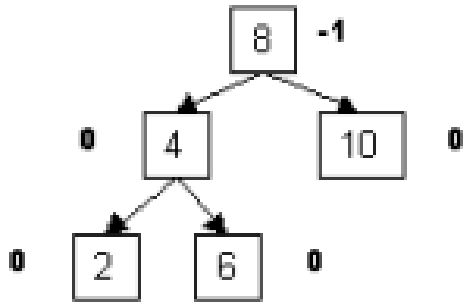


Identificamos a classe 2, que demanda uma rotação dupla à direita.

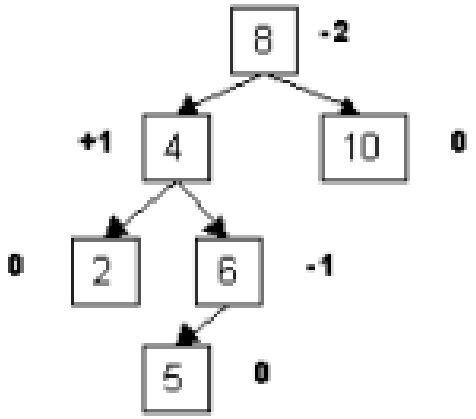
Iniciamos por uma rotação no nó filho com FB +1, no caso, com uma rotação à esquerda devido ao sinal +. Seguindo por uma rotação à direita no nó com FB -2.

O que gera a sequência de árvores a seguir.

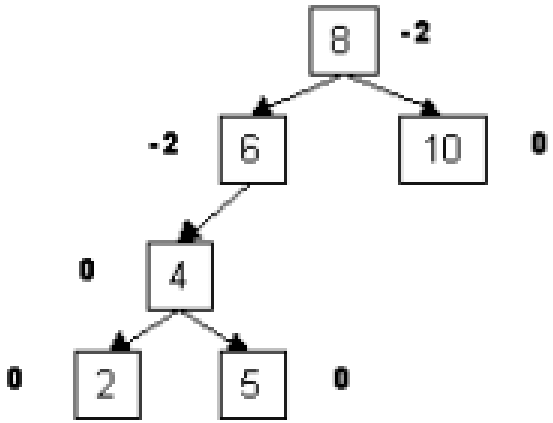
Árvore AVL



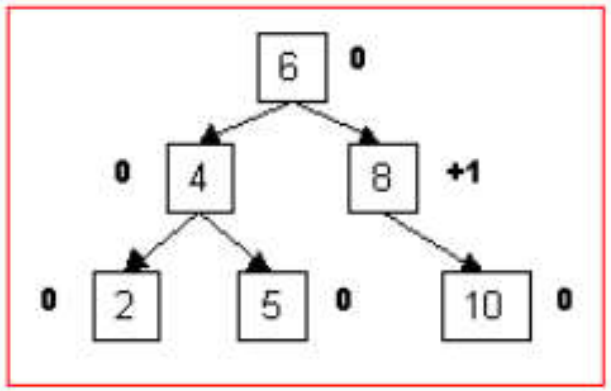
Árvore inicial



Árvore desbalanceada (após inserção do valor 5)



Árvore em processo de balanceamento (após a 1ª rotação)



Árvore balanceada (após a 2ª rotação)

Árvore AVL

Descreva uma sequência de passos para a construção de uma árvore AVL.

1. insira o novo nó normalmente, ou seja, da mesma maneira que insere-se um nó em uma árvore binária de busca;

2. iniciando com o nó pai do nó recém inserido, teste se a propriedade AVL foi violada, ou seja, atualize e teste se algum dos FB's passou a ser 2 ou -2. Temos duas possibilidades:

2.1. A condição AVL foi violada

2.1.1. Execute a operação de rotação conforme o caso (tipo 1 ou tipo 2);

2.1.2. Volte ao passo 1;

2.2. A condição AVL não foi violada, volte ao passo 1;

Árvore AVL

Com base no que foi visto, proponha uma estrutura para um nó de uma árvore AVL implementada dinamicamente. Considere que a informação armazenada em cada nó da árvore resumi-se a um valor inteiro.

```
typedef struct nodo
{
    int num, altd, alte;
    struct nodo *dir, *esq;
}NODO;
```

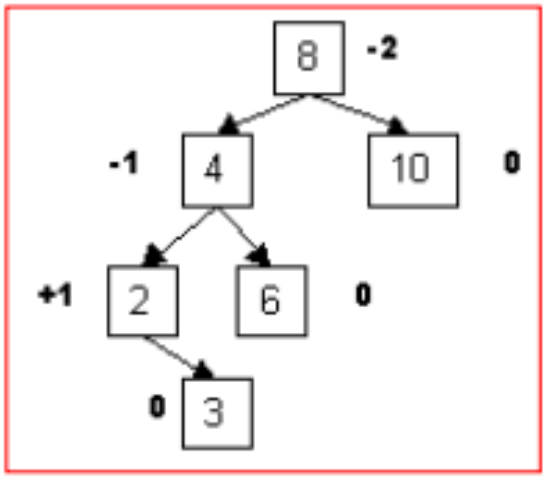
Como seria definido o tipo árvore AVL?

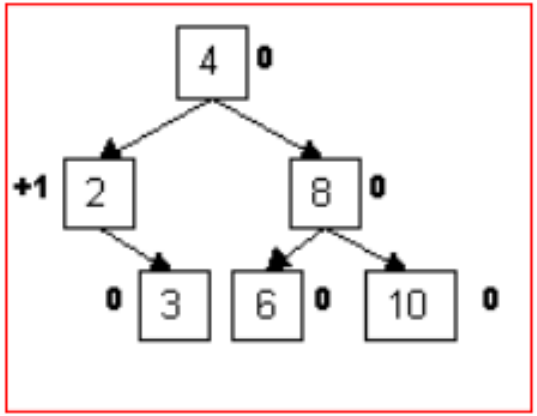
```
typedef NODO * ArvoreAVL;
```

Árvore AVL

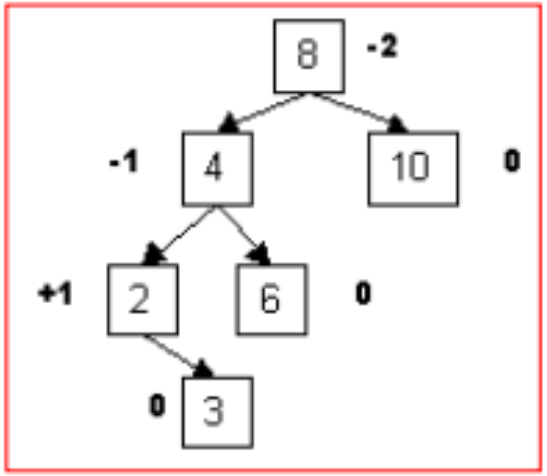
Com base no que foi visto, implemente a operação de rotação à direita sobre o nó recebido como parâmetro. Considere o protótipo abaixo para a função que implementará a operação em questão.

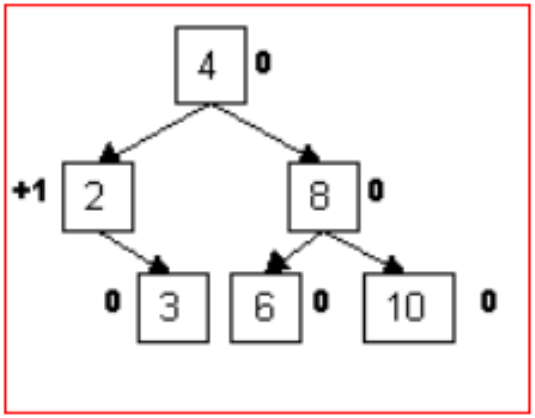
void rotacao_direita(ArvoreAVL *arvore);





```
void rotacao_direita(ArvoreAVL *arvore)
{
  ArvoreAVL aux1, aux2;
  aux1 = (*arvore)->esq;
  aux2 = aux1->dir;
  (*arvore)->esq = aux2;
  aux1->dir = (*arvore);
  if ((*arvore)->esq == NULL)
    (*arvore)->alte = 0;
  else
    if ((*arvore)->esq->alte > (*arvore)->esq->altd)
      (*arvore)->alte = (*arvore)->esq->alte+1;
    else
      (*arvore)->alte = (*arvore)->esq->altd+1;
  if (aux1->dir->alte > aux1->dir->altd)
    aux1->altd = aux1->dir->alte + 1;
  else
    aux1->altd = aux1->dir->altd + 1;
  *arvore = aux1;
}
```

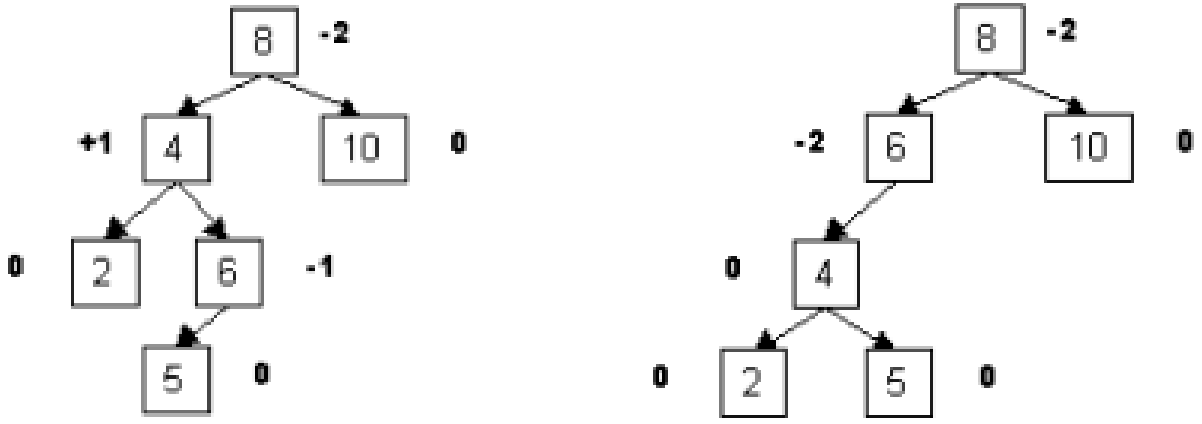




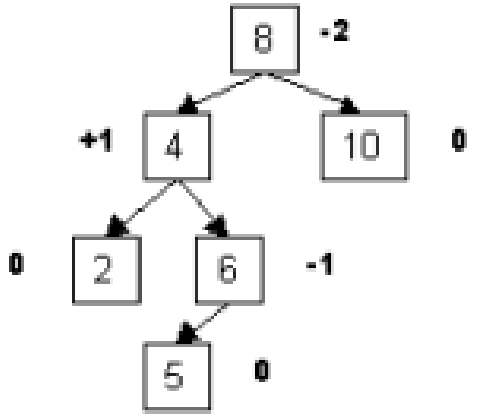
Árvore AVL

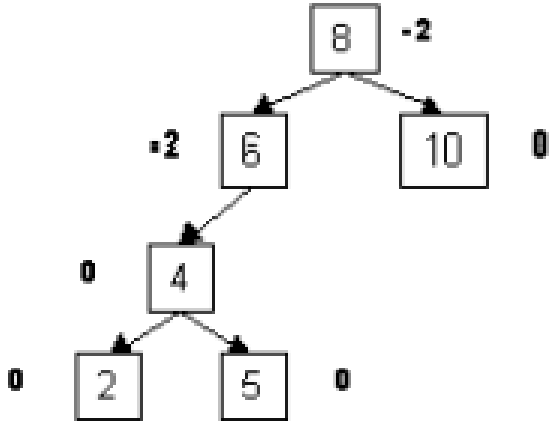
Com base no que foi visto, implemente a operação de rotação à esquerda sobre o nó recebido como parâmetro. Considere o protótipo abaixo para a função que implementará a operação em questão.

void rotacao_esquerda(ArvoreAVL *arvore);



```
void rotacao_esquerda(ArvoreAVL *arvore)
{
  ArvoreAVL aux1, aux2;
  aux1 = (*arvore)->dir;
  aux2 = aux1->esq;
  (*arvore)->dir = aux2;
  aux1->esq = (*arvore);
  if ((*arvore)->dir == NULL)
    (*arvore)->altd = 0;
  else
    if ((*arvore)->dir->alte > (*arvore)->dir->altd)
      (*arvore)->altd = (*arvore)->dir->alte+1;
    else
      (*arvore)->altd = (*arvore)->dir->altd+1;
  if (aux1->esq->alte > aux1->esq->altd)
    aux1->alte = aux1->esq->alte + 1;
  else
    aux1->alte = aux1->esq->altd + 1;
  *arvore = aux1;
}
```





Árvore AVL

Com base nas operações de rotação à esquerda e à direita, implemente a operação de balanceamento sobre o nó recebido como parâmetro. Considere o protótipo abaixo para a função que implementará a operação em questão.

void balanceamento (ArvoreAVL *arvore);

```
void balanceamento(ArvoreAVL *arvore) {
   int d, df;
   d = (*arvore)->altd - (*arvore)->alte;
   if (d == 2){
      df = (*arvore)->dir->altd - (*arvore)->dir->alte;
      if (df >= 0)
         rotacao_esquerda(arvore);
      else {
         rotacao_direita(&((*arvore)->dir));
         rotacao_esquerda(arvore);
      }
   }
   else
      if (d == -2) {
         df = (*arvore)->esq->altd - (*arvore)->esq->alte;
         if (df <= 0)
            rotacao_direita(arvore);
         else{
            rotacao_esquerda(&((*arvore)->esq));
            rotacao_direita(arvore);
         }
      }
}
```

Árvore multivias

Vimos inicialmente um conceito mais amplo de árvore e depois o restringimos fixando o número máximo de filhos que um nó pode ter em dois.

Se permitirmos a determinação de mais itens de dados e mais filhos por nó teremos como resultado árvores denominadas **Multivias** ou **M-vias**.

Uma estrutura multivia com algoritmo eficiente deve considerar :

- Tempo de acesso a cada nó
- Balanceamento da árvore

UNIVASF

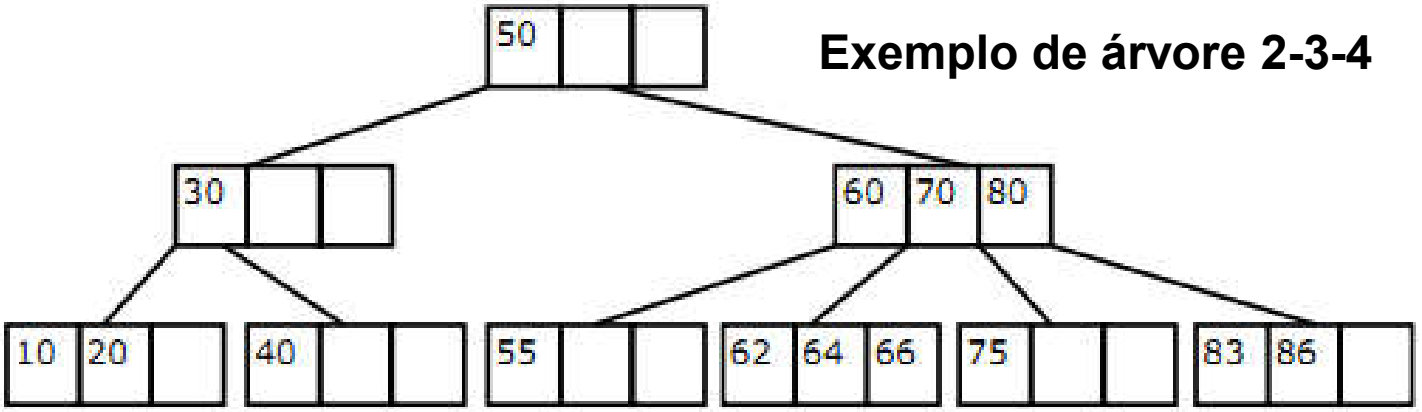
Árvore 2-3-4

Um bom exemplo de uma árvore multivia é a árvore 2-3-4.

Uma árvore 2-3-4 pode ter até quatro filhos e três itens de dados por nó.

Razões para se estudar árvores 2-3-4:

- São árvores balanceadas;
- São fáceis de implementar;
- Servem como uma introdução para o estudo de árvores B.



Exemplo de árvore 2-3-4

Árvore 2-3-4

Em uma árvore 2-3-4 cada nó pode conter um, dois ou três itens de dados.

Um nó interno deve sempre ter um filho a mais que seus itens de dados.

Devido a uma árvore 2-3-4 possuir nós com até quatro filhos, ela é chamada de árvore multivias de ordem 4.

Por que o 2, 3 e 4 no nome da árvore 2-3-4?

Porque estas são as quantidades possíveis de filhos que um dado nó não folha pode ter.

UNIVASF

Árvore 2-3-4

Em uma árvore binária, todos os filhos com chaves menores que a chave do nó estão “enraizados” no nó filho à esquerda, e todos os filhos com chaves maiores estão “enraizados” no nó filho à direita.

Na árvore 2-3-4 o princípio é o mesmo, com alguns detalhes a mais:

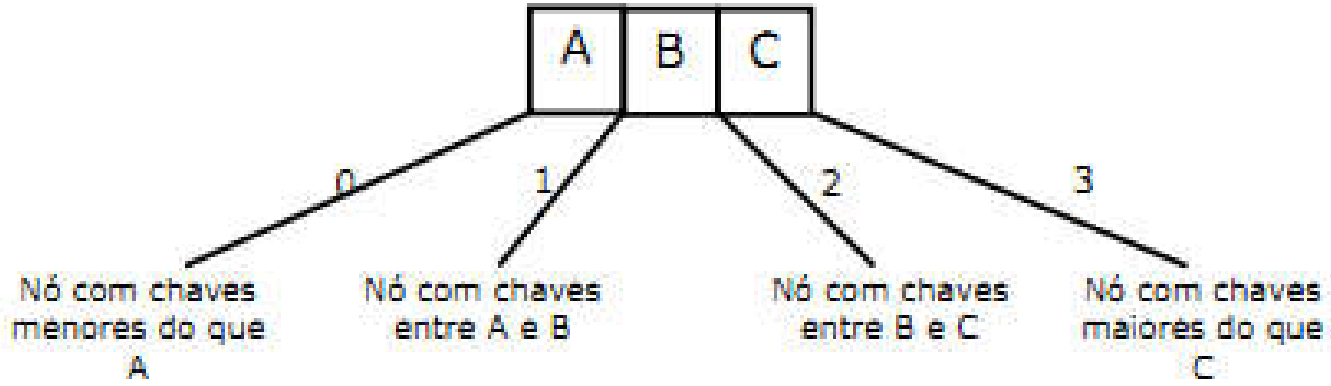
- todos os itens de dados na subárvore “enraizada” no filho 0, possuem valores menores que o da chave 0;

- todos os itens de dados na subárvore “enraizada” no filho 1, possuem valores maiores do que o da chave 0, mas menores do que a chave 1;

Árvore 2-3-4

- todos os itens de dados na subárvore “enraizada” no filho 2, possuem valores maiores do que o da chave 1, mas menores do que a chave 2;

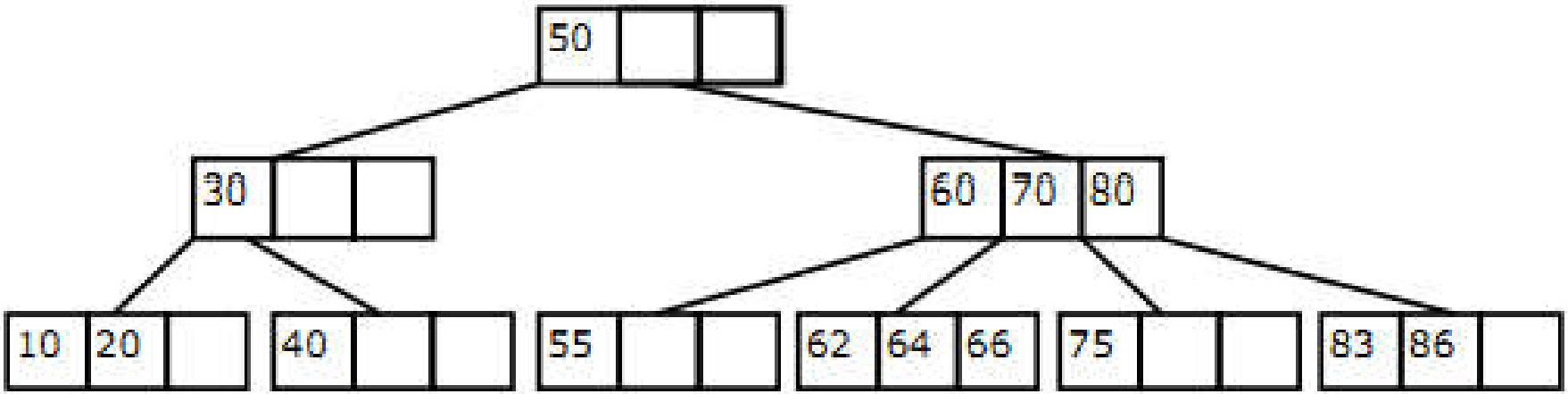
- todos os itens de dados na subárvore “enraizada” no filho 3, possuem valores maiores do que o da chave 2.



Valores duplicados geralmente não são permitidos, o que possibilita que não nos preocupemos com comparações de chaves iguais.

Árvore 2-3-4

Veremos agora um exemplo de como ocorre a pesquisa por uma chave em uma árvore 2-3-4. Simularemos a busca pela chave 64.



Iniciamos a busca pela raiz e ao não encontrar a chave percebemos que a chave é maior que 50 e portanto a busca segue no filho 1, o qual também não contém a chave em seus itens de dados. Como 64 está entre 60 e 70 a busca segue novamente para o filho 1. Desta vez a chave é localizada.

Árvore 2-3-4

Trataremos agora do processo de inserção de uma nova chave em uma árvore 2-3-4.

Novos itens de dados são sempre inseridos nas folhas. Por quê?

Porque se os itens forem inseridos em um nó com filhos, então o número de filhos necessitará ser mudado para manter a estrutura da árvore, a qual estipula que a árvore deve ter um filho a mais do que os itens de dados em cada nó não folha.

O processo de inserção começa pela busca do nó folha apropriado.

Se apenas nós que não estão cheios são encontrados durante a busca, a inserção é mais fácil.

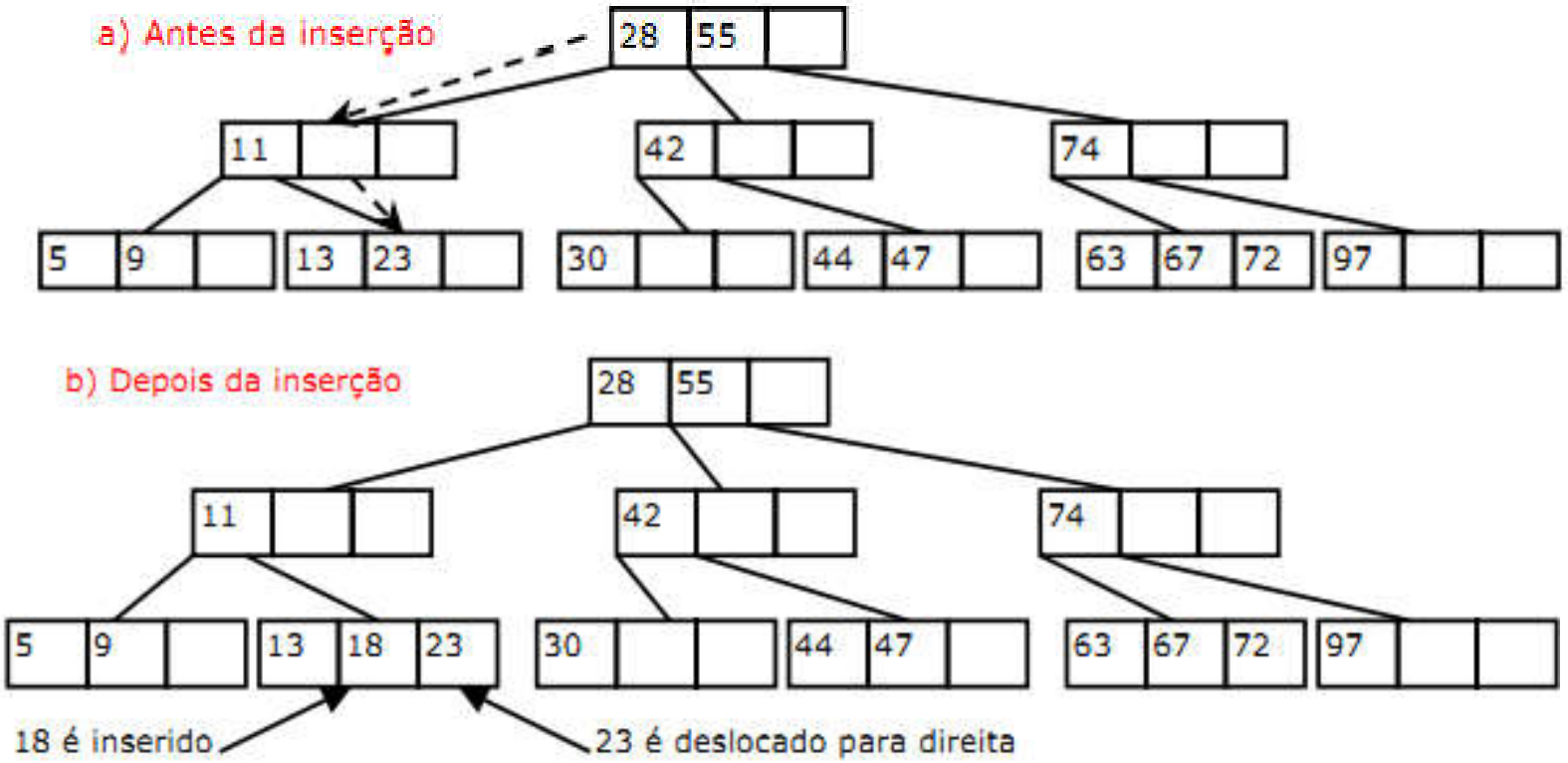
UNIVASF

Árvore 2-3-4

A inserção pode envolver a movimentação de um ou dois itens de dados em um nó.

As chaves deverão estar na ordem correta após o novo item ser inserido.

Exemplo da inserção item 18 na árvore abaixo:



Árvore 2-3-4

As inserções tornam-se mais complicadas se um nó cheio é encontrado no caminho abaixo do ponto de inserção.

Quando isso ocorre o nó precisa ser dividido.

É este processo de divisão que mantém a árvore balanceada.

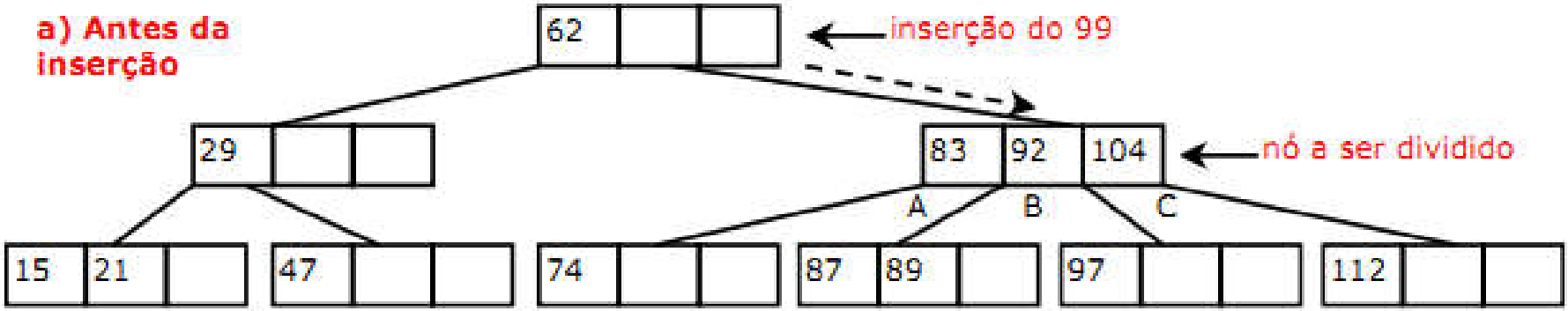
O tipo de árvore 2-3-4 que estamos estudando é frequentemente chamado de árvore 2-3-4 top-down, porque os nós são divididos “para baixo” do ponto de inserção.

Chamaremos os itens de dados a serem divididos de A, B e C.

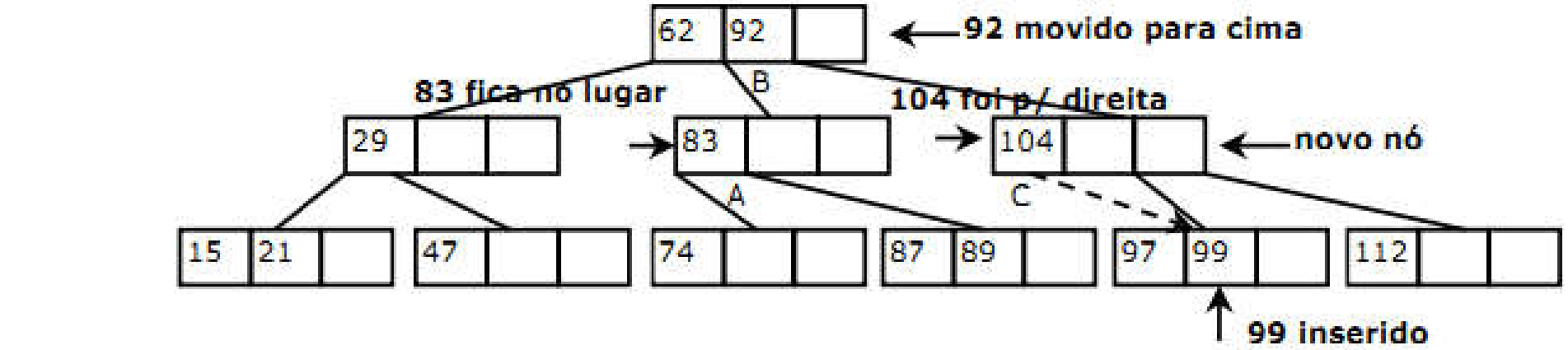
Assumiremos que o nó a ser dividido não é a raiz, examinaremos a divisão do nó raiz depois.

UNIVASF

Árvore 2-3-4



- Um novo nó vazio é criado. Ele é parente (*sibling*) do nó que está sendo dividido, e é colocado a sua direita;
- O item de dado C é movido para o novo nó;
- O item de dado B é movido para o pai do nó que está sendo dividido;
- O item de dado A fica aonde ele está;
- Os dois filhos mais à direita são desconectados do nó que está sendo dividido e são conectados no novo nó.



Árvore 2-3-4

Quando uma raiz cheia é encontrada no início da busca para encontrar o ponto de inserção, o processo de inserção é ligeiramente mais complicado:

Um novo nó é criado, tornando-se a nova raiz, e a antiga raiz é dividida criando um novo nó irmão;

O item de dado C é movido para o novo nó irmão da antiga raiz;

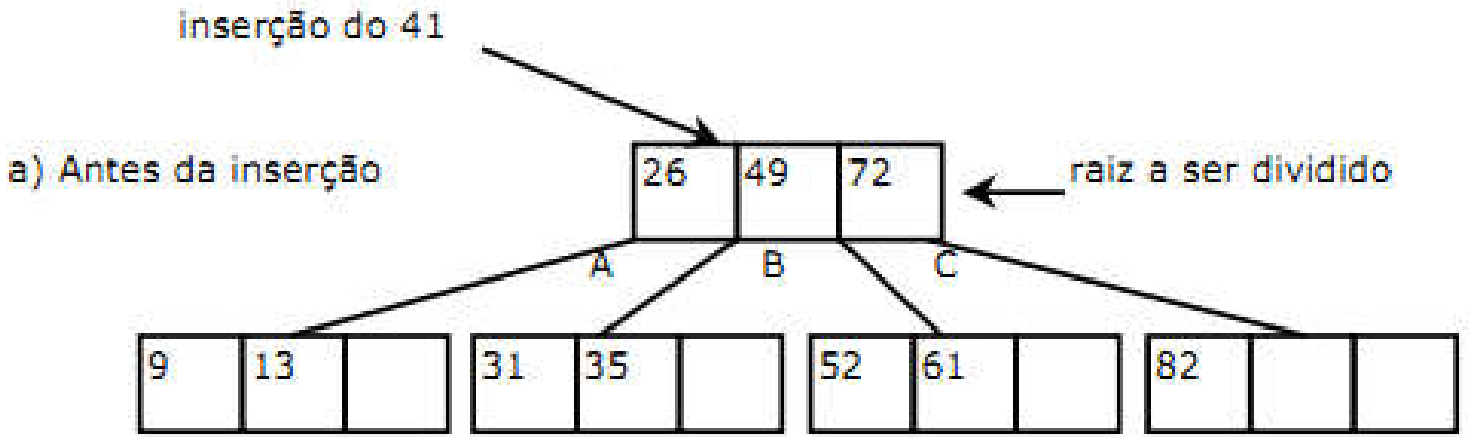
O item de dado B é movido para a nova raiz;

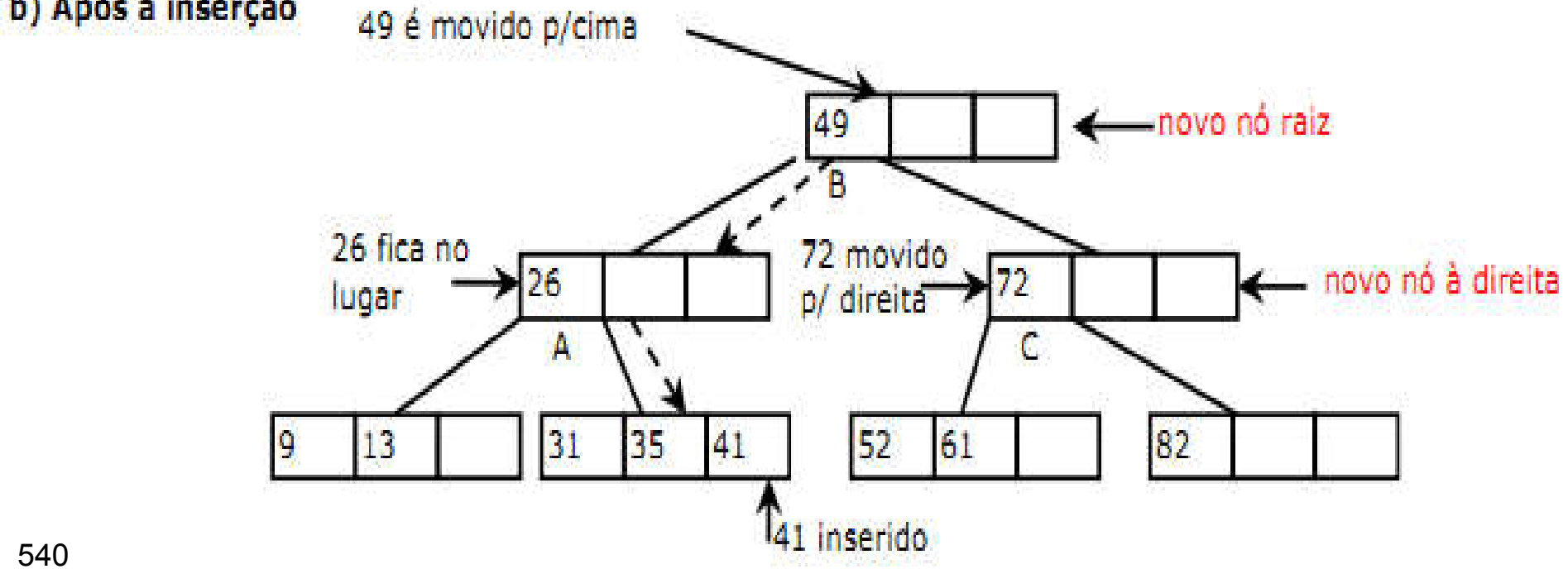
O item de dados A é deixado onde está;

Os dois filhos mais a direita do nó que está sendo dividido são desconectados dele e conectados no novo nó do lado direito.

UNIVASF

Árvore 2-3-4


inserção do 41
a) Antes da inserção
26
49
72
raiz a ser dividido
A
B
C
9
13
31
35
52
61
82


b) Após a inserção
49 é movido p/cima
49
novo nó raiz
B
26 fica no lugar
26
72 movido p/ direita
72
novo nó à direita
A
C
9
13
31
35
41
52
61
82
41 inserido

FIM!