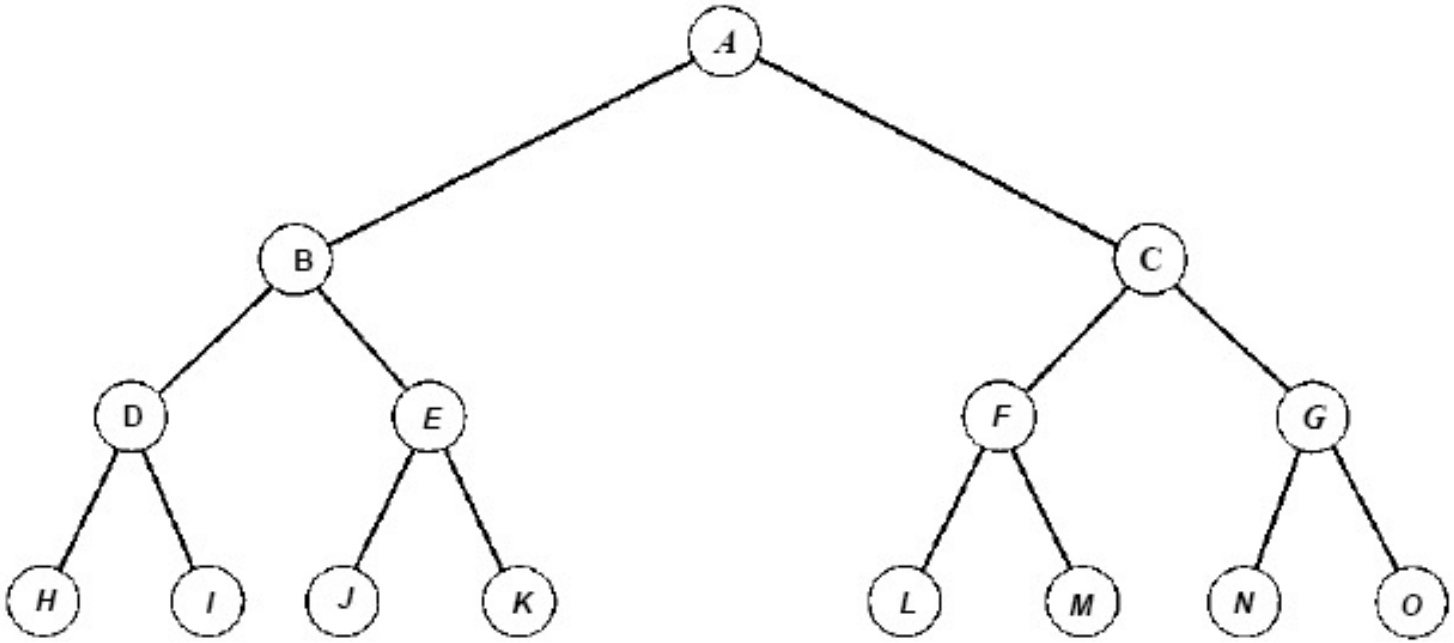
ADT – Árvores Binárias – Definição

Embora as árvores naturais cresçam com suas raízes fincadas na terra e suas folhas no ar, os cientistas de computação retratam as estruturas de dados em árvore com a raiz no topo e as folhas no chão.
O sentido da raiz para as folhas é "para baixo" e o sentido oposto é "para cima". Quando percorre-se uma árvore a partir das folhas na direção da raiz, diz-se que está "subindo" a árvore, e se partir da raiz para as folhas, está "descendo" a árvore.

ADT - Árvores

ADT - Árvores



Fonte: TENENBAUM et.al., 1999, p. 308.

ADT – Árvores – Definição

Árvore é uma estrutura de dados não-linear e bidimensional.

Tipo abstrato de dados (ADT).

Uma árvore consiste em nós conectados por *arestas* (linhas).
Os nós são representados como círculos e as arestas como linhas, conectando os círculos.
As arestas devem ser representadas em um programa por referências.
Se todo nó poderá ter no máximo dois filhos, a árvore será chamada de *árvore binária*.

ADT – Árvores Binárias – Definição

Árvore binária é definida por um conjunto finito de elementos que está vazio ou é particionado em três subconjuntos disjuntos.

O primeiro subconjunto contém um único elemento, chamado **raiz** da árvore.

Os outros dois subconjuntos são em si mesmo árvores binárias, chamadas **subárvores esquerda** e **direita** da árvore original.

Uma subárvore esquerda ou direita pode estar vazia.

Cada elemento de uma árvore binária é chamado **nó** da árvore.

ADT – Árvores Binárias – Definição

Um método convencional de ilustrar uma árvore binária aparece na Figura 1.

Essa árvore consiste em nove nós, com A como sua raiz.

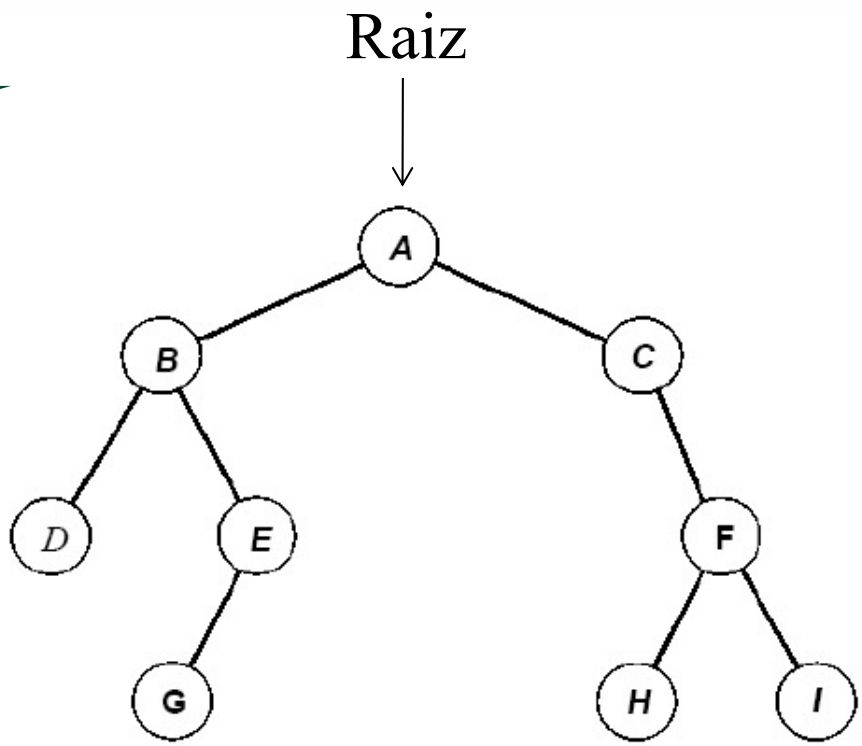


Figura 1 – Árvore binária.
Fonte: TENENBAUM et.al., 1999, p. 304.

ADT – Árvores Binárias – Definição

Um método convencional de ilustrar uma árvore binária aparece na Figura 1.

Essa árvore consiste em nove nós, com A como sua raiz.

Sua subárvore esquerda está enraizada em B e sua subárvore direita, em C. Isso aparece indicado pelas duas ramificações que saem de A: para B, no lado esquerdo e para C, no lado direito.

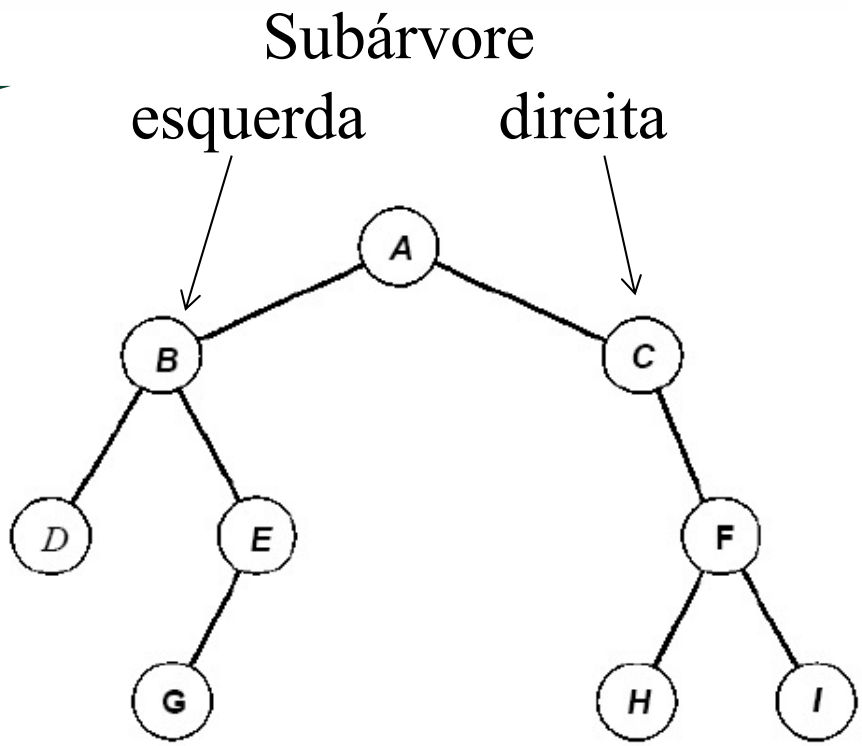


Figura 1 – Árvore binária.
Fonte: TENENBAUM et.al., 1999, p. 304.

ADT – Árvores Binárias – Definição

Um método convencional de ilustrar uma árvore binária aparece na Figura 1.

Essa árvore consiste em nove nós, com A como sua raiz. Sua subárvore esquerda está enraizada em B e sua subárvore direita, em C. Isso aparece indicado pelas duas ramificações que saem de A: para B, no lado esquerdo e para C, no lado direito.

Por exemplo, a subárvore esquerda, da árvore binária enraizada em C e a subárvore direita da árvore binária enraizada em E estão, ambas, vazias. As árvores binárias enraizadas em D, G, H e I têm subárvores direita e esquerda vazias.

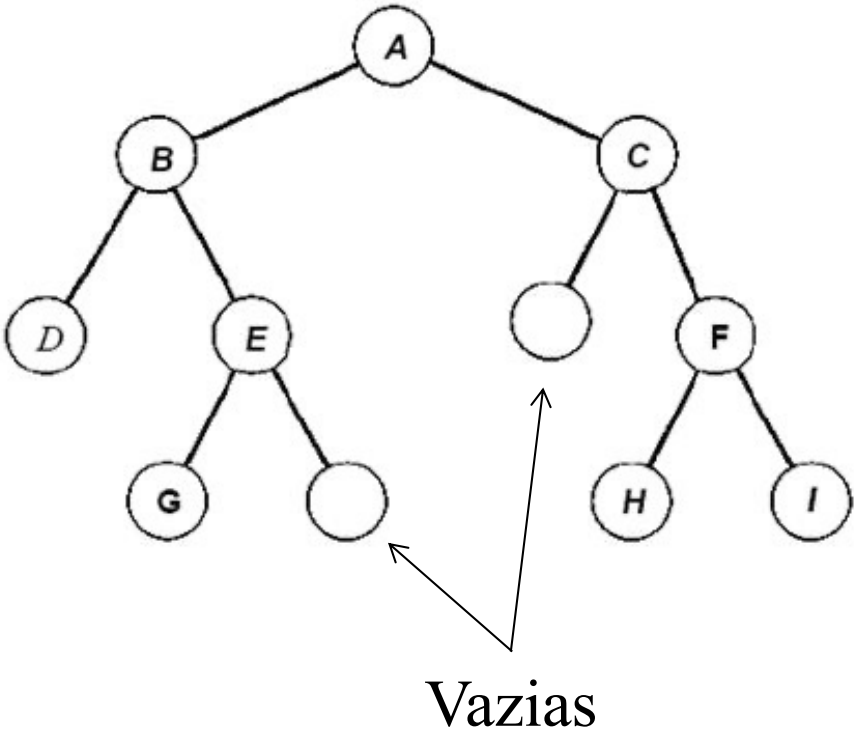


Figura 1 – Árvore binária.
Fonte: TENENBAUM et.al., 1999, p. 304.

ADT – Árvores Binárias – Definição

A Figura 2 ilustra algumas estruturas que não são árvores binárias. Procure entender por que cada uma delas não é uma árvore binária conforme definição anterior.

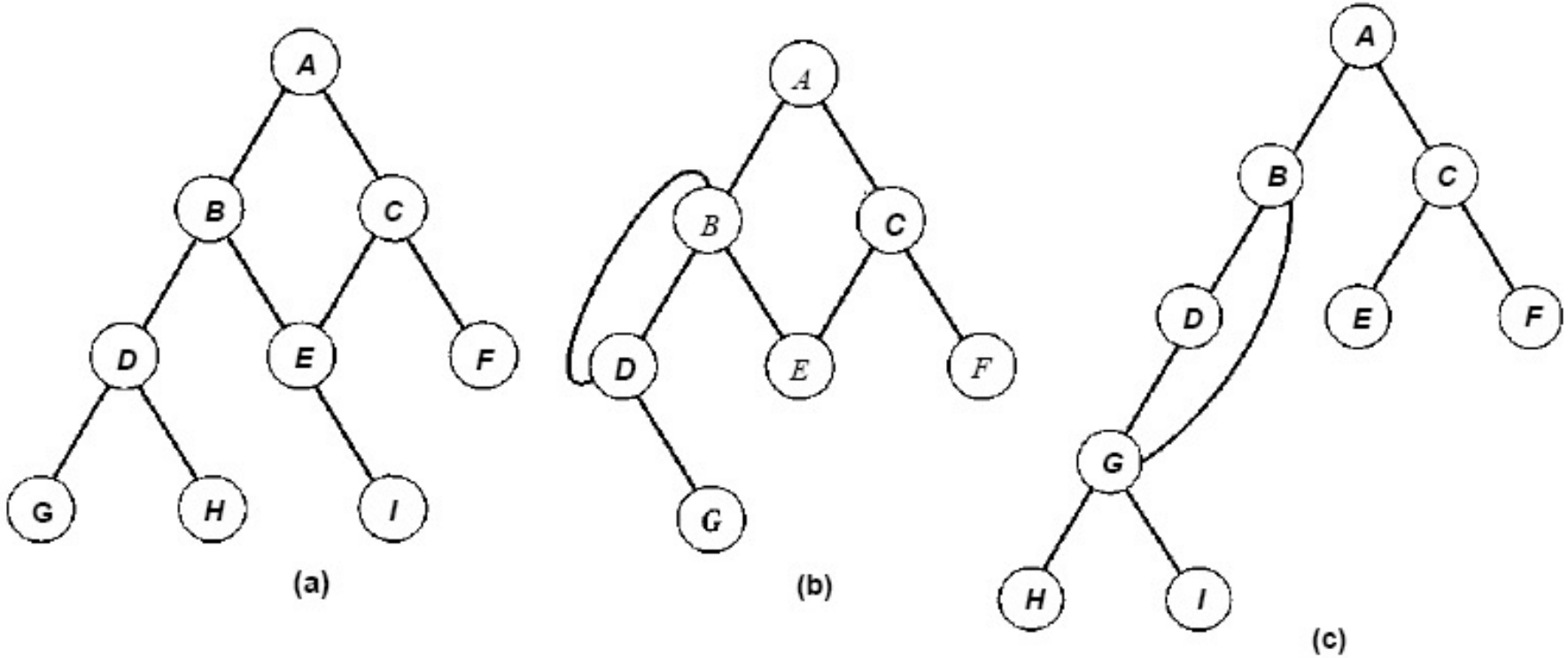


Figura 2 – Estruturas que não são árvores binárias.
Fonte: TENENBAUM et.al., 1999, p. 305.

ADT – Árvores Binárias – Definição

Se A é a raiz de uma árvore binária e B é a raiz de sua subárvore direita ou esquerda, então diz-se que A é o pai de B e que B é o filho direito ou esquerdo de A.

ADT – Árvores Binárias – Definição

Se todo nó que não é folha - nó que não possui filhos - numa árvore binária tiver subárvores esquerda e direita não-vazias, a árvore será considerada uma **árvore estritamente binária**. Sendo assim, a árvore da Figura 3 é estritamente binária, enquanto a da Figura 1 não é (porque os nós C e E têm um filho cada).

Uma árvore estritamente binária com n folhas contém sempre **2*n – 1** nós (n é equivalente ao número de folhas).
A comprovação desse fato será deixada como exercício.

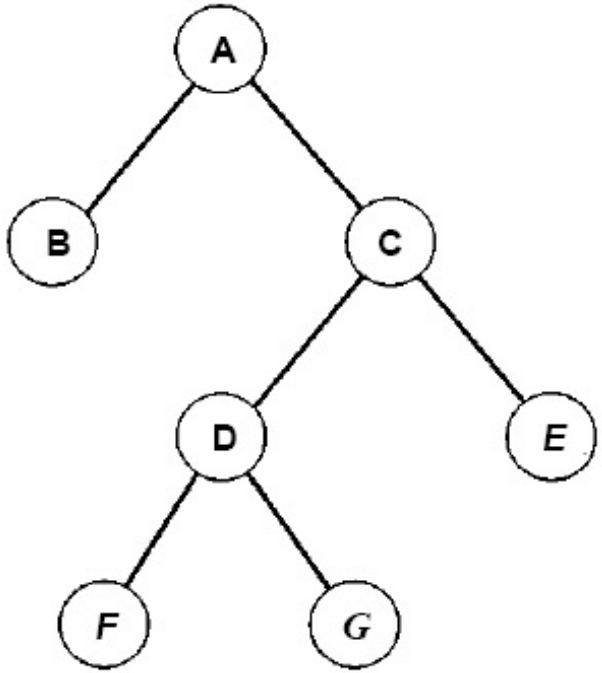
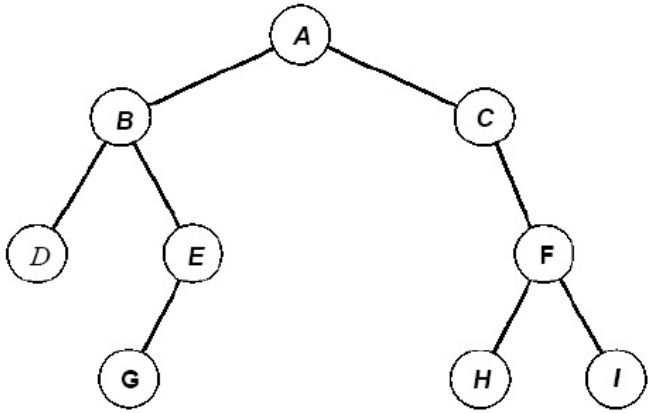


Figura 3 – Árvore estritamente binária.
Fonte: TENENBAUM et.al., 1999, p. 304.

ADT – Árvores Binárias – Definição

O nível de um nó numa árvore binária é definido como segue: a raiz da árvore tem nível 0, e o nível de qualquer outro nó na árvore é um nível a mais que o nível de seu pai.

Por exemplo, na árvore binária da Figura 1 (abaixo), o nó E está no nível 2 e o nó H, no nível 3. A profundidade de uma árvore binária significa o nível máximo de qualquer folha na árvore. Isso equivale ao tamanho do percurso mais distante da raiz até qualquer folha. Sendo assim, a profundidade da árvore da Figura 1 é 3.



ADT – Árvores Binárias – Definição

Uma árvore completa é uma árvore em que todos os nós com menos de dois filhos ficam no último e no penúltimo nível (Figura 4).

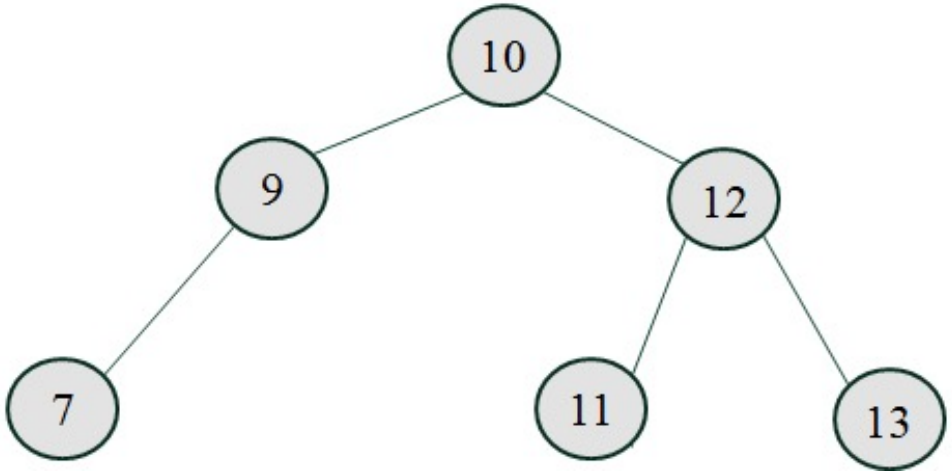


Figura 4 – Árvore completa.
Fonte: O autor.

ADT – Árvores Binárias – Definição

Uma árvore binária cheia é uma árvore em que se um nó tem alguma subárvore vazia, então ele está no último nível (Figura 5).
Portanto, **toda árvore cheia é completa e estritamente binária**.

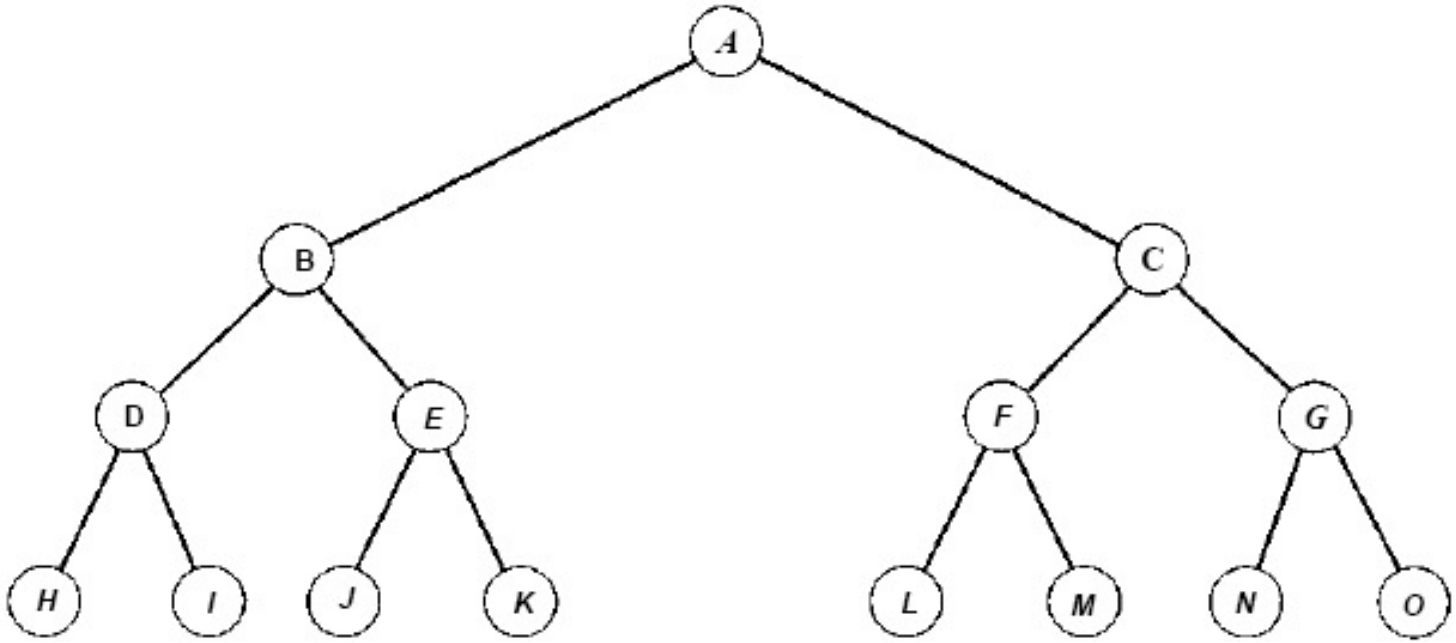


Figura 5 – Árvore binária cheia de nível 3.
Fonte: TENENBAUM *et al.*, 1999, p. 308.

ADT – Árvores Binárias – Operações

Existem operações primitivas que podem ser aplicadas a uma árvore binária.

Se *p* é uma referência para um nó *nd* de uma árvore binária, a função ***info****(p)* retorna o conteúdo de *nd*.

As funções ***left****(p)*, ***right****(p)*, ***father****(p)* e ***brother****(p)* retornam referências para o filho esquerdo de *nd*, filho direito de *nd*, pai de *nd* e irmão de *nd*, respectivamente. Essas funções retornarão a referência *null* se *nd* não tiver filho esquerdo, filho direito, pai ou irmão.

Finalmente, as funções lógicas ***isleft****(p)* e ***isright****(p)* retornam o valor *true* se *nd* for um filho esquerdo ou direito, respectivamente, de algum outro nó na árvore, e *false*, caso contrário.

ADT – Árvores Binárias – Operações

Observe que as funções *isleft(p)*, *isright(p)* e *brother(p)* podem ser implementadas usando-se as funções *left(p)*, *right(p)* e *father(p)*. Por exemplo, ***isleft****(p)* pode ser implementado assim:

```
q = father(p)
if q is None:
   return False          /* p aponta para a raiz */
if left(q) is p:
   return True
return False
```

ou, ainda mais simples, como *father(p) and p == left(father(p))*.

isright pode ser implementado de modo semelhante.

ADT – Árvores Binárias – Operações

brother(p) pode ser implementado usando-se *isleft* ou *isright*, como segue:

```
if father(p) is None:
   return None              /* p aponta para a raiz */
if isleft(p):
   return right(father(p))
return left(father(p))
```

ADT – Árvores Binárias – Operações

Ao construir uma árvore binária, as operações *maketree*, *setleft* e *setright* são úteis, *maketree(x)* cria uma nova árvore binária consistindo num único nó com o campo de informação *x* e retorna uma referência para esse nó.

setleft(p,x) aceita uma referência *p* para um nó de uma árvore binária sem filhos. Ele cria um novo filho esquerdo de *node(p)* com o campo de informação *x*.

setright(p,x) é análoga a *setleft*, exceto pelo fato de que ela cria um filho direito de *node(p)*.

ADT – Árvores Binárias – Aplicação

Uma árvore binária é uma estrutura de dados útil quando precisam ser tomadas decisões bidirecionais em cada ponto de um processo.

Por exemplo, suponha que se precisa encontrar as repetições numa lista de números.

Uma maneira de fazer isso é comparar cada número com todos os que o precedem.

Entretanto, isso envolve um grande número de comparações.

ADT – Árvores Binárias – Aplicação

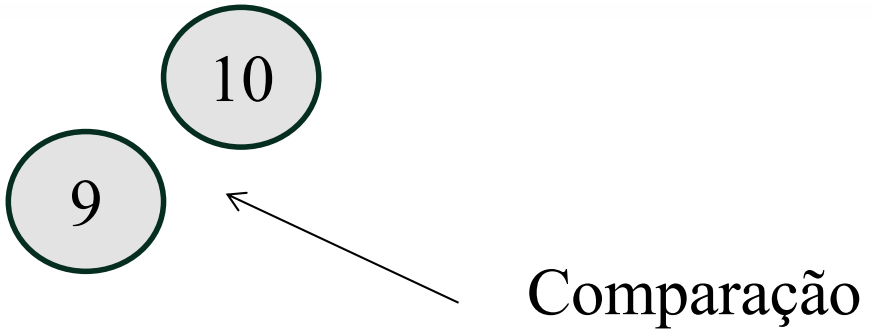
O número de comparações pode ser reduzido usando-se uma árvore binária.
O primeiro número na lista é colocado num nó estabelecido como a raiz de uma árvore binária com as subárvores esquerda e direita vazias.
Cada número sucessivo na lista é, então, comparado ao número na raiz. Se coincidirem, tem-se uma repetição. Se for menor, se examina a subárvore esquerda; se for maior, examina-se a subárvore direita. Se a subárvore estiver vazia, o número não será repetido e será colocado num novo nó nesta posição na árvore.
Se a subárvore não estiver vazia, compara-se o número ao conteúdo da raiz da subárvore e o processo inteiro será repetido com a subárvore.

ADT – Árvores Binárias – Aplicação

10

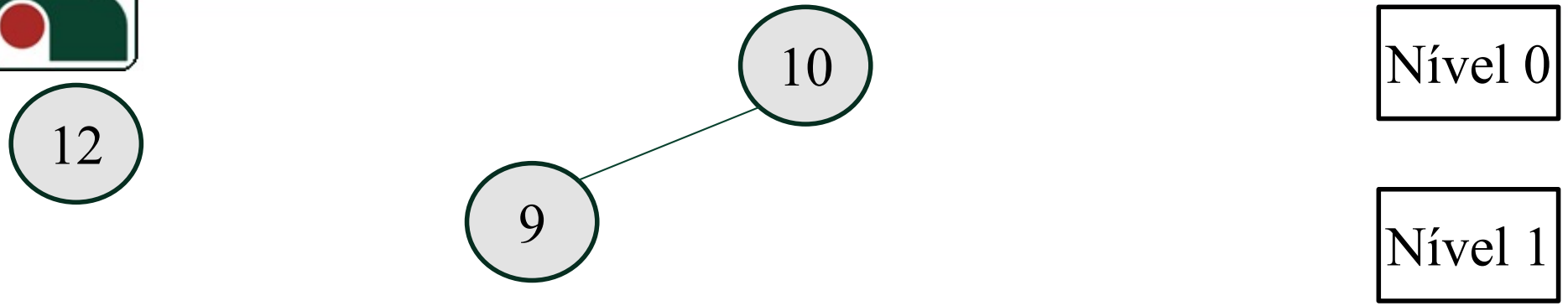
ADT – Árvores Binárias – Aplicação

ADT – Árvores Binárias – Aplicação

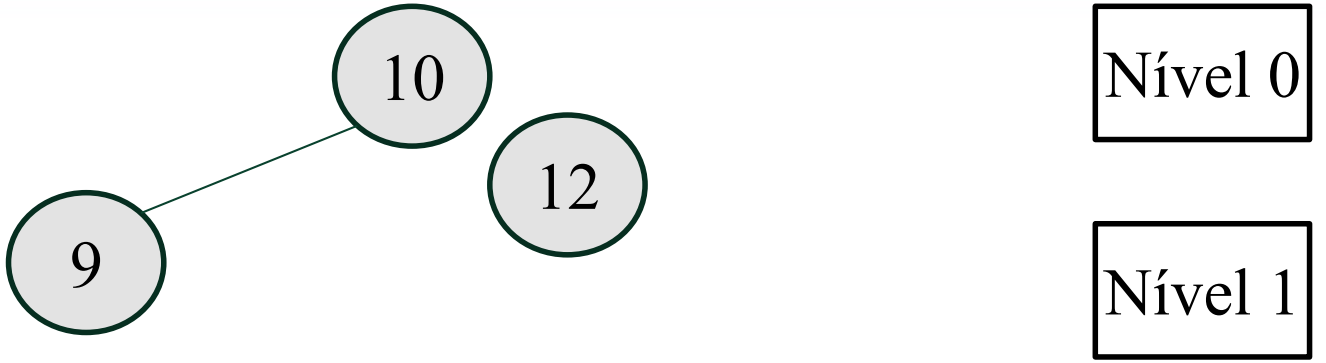


ADT – Árvores Binárias – Aplicação

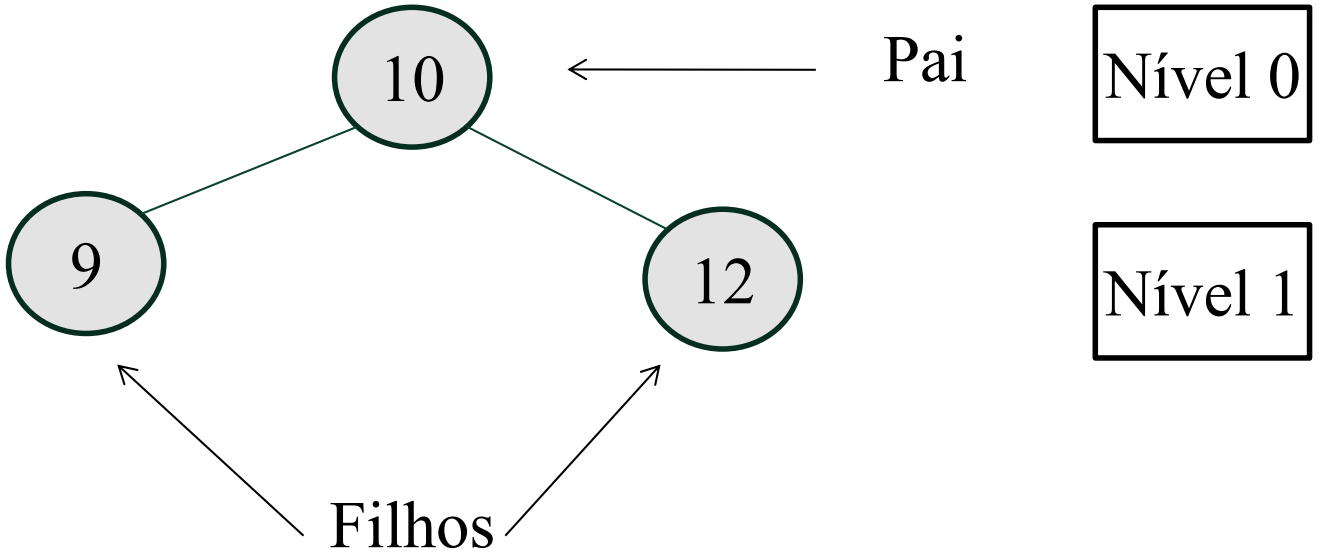
ADT – Árvores Binárias – Aplicação



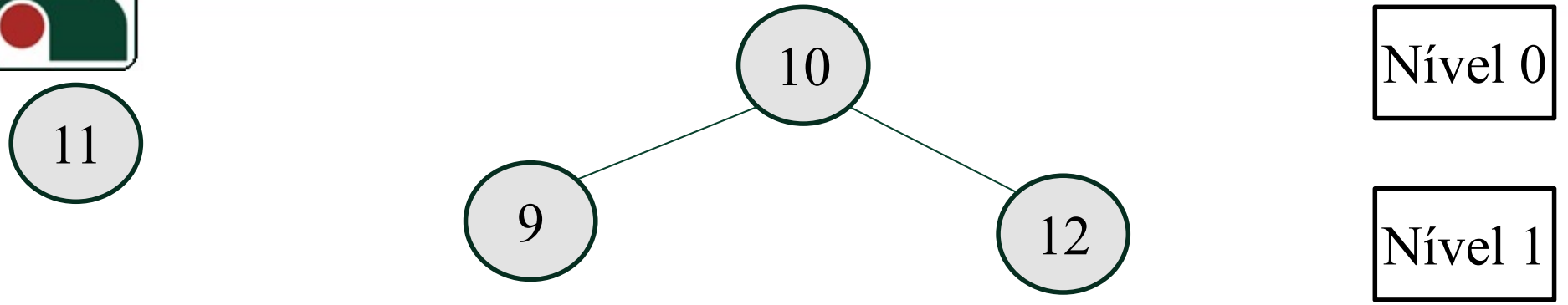
ADT – Árvores Binárias – Aplicação



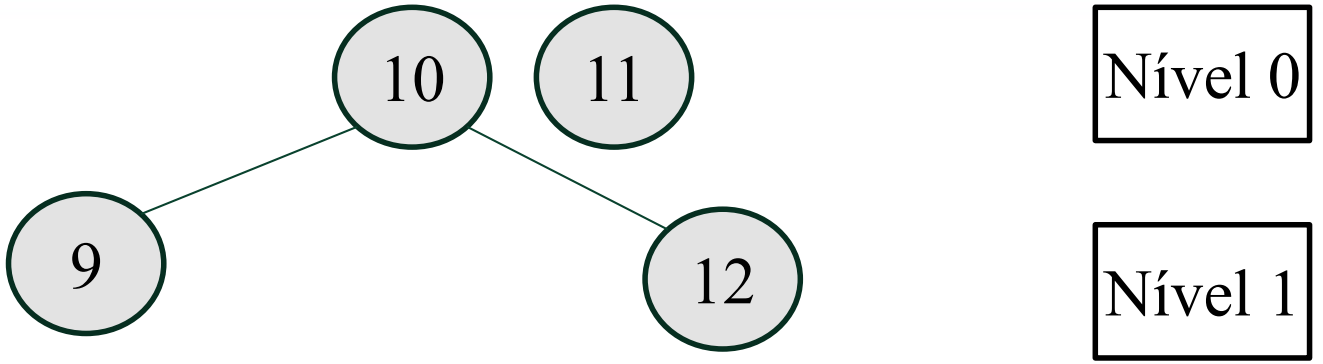
ADT – Árvores Binárias – Aplicação



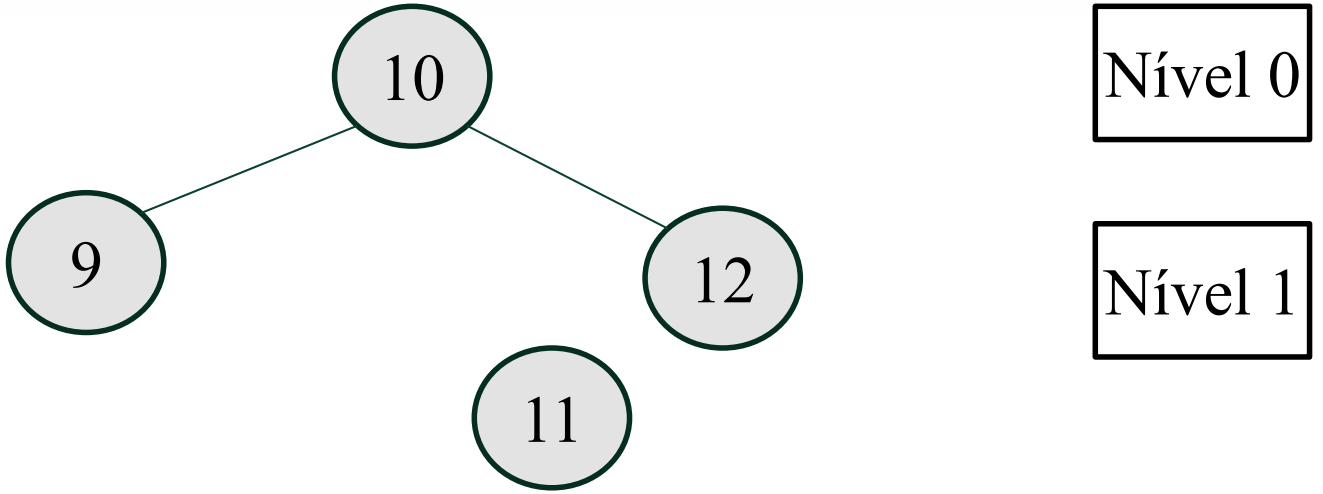
ADT – Árvores Binárias – Aplicação



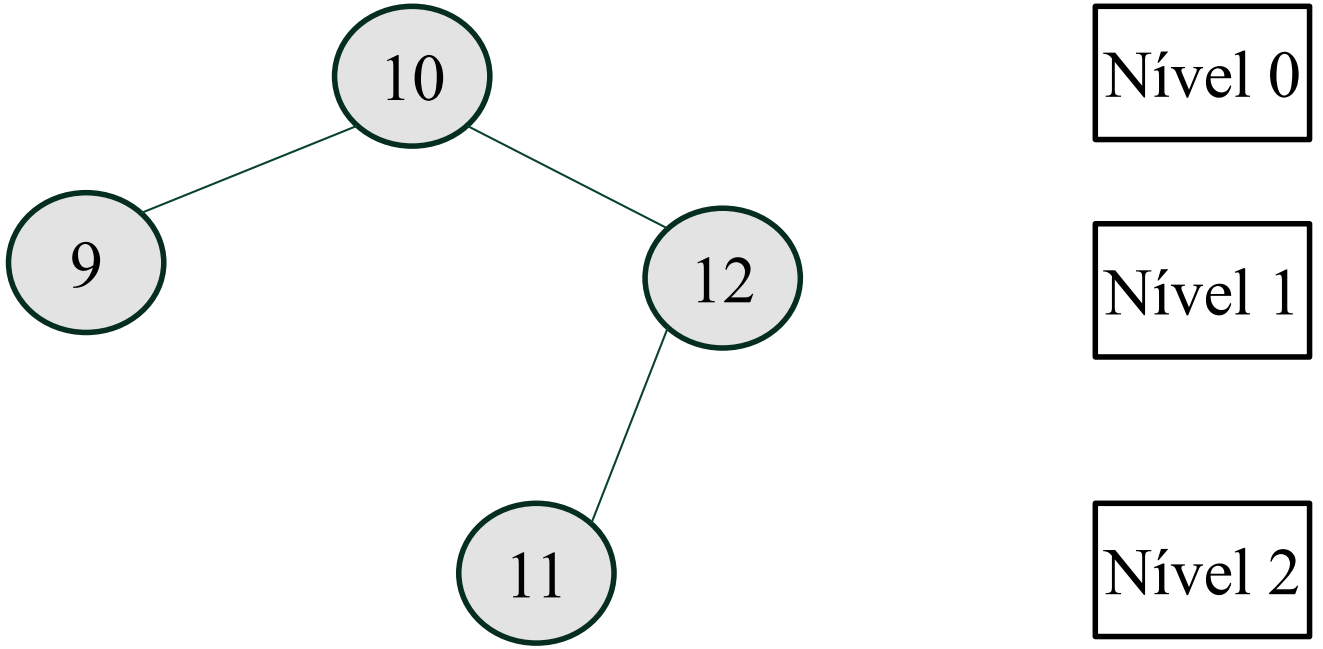
ADT – Árvores Binárias – Aplicação



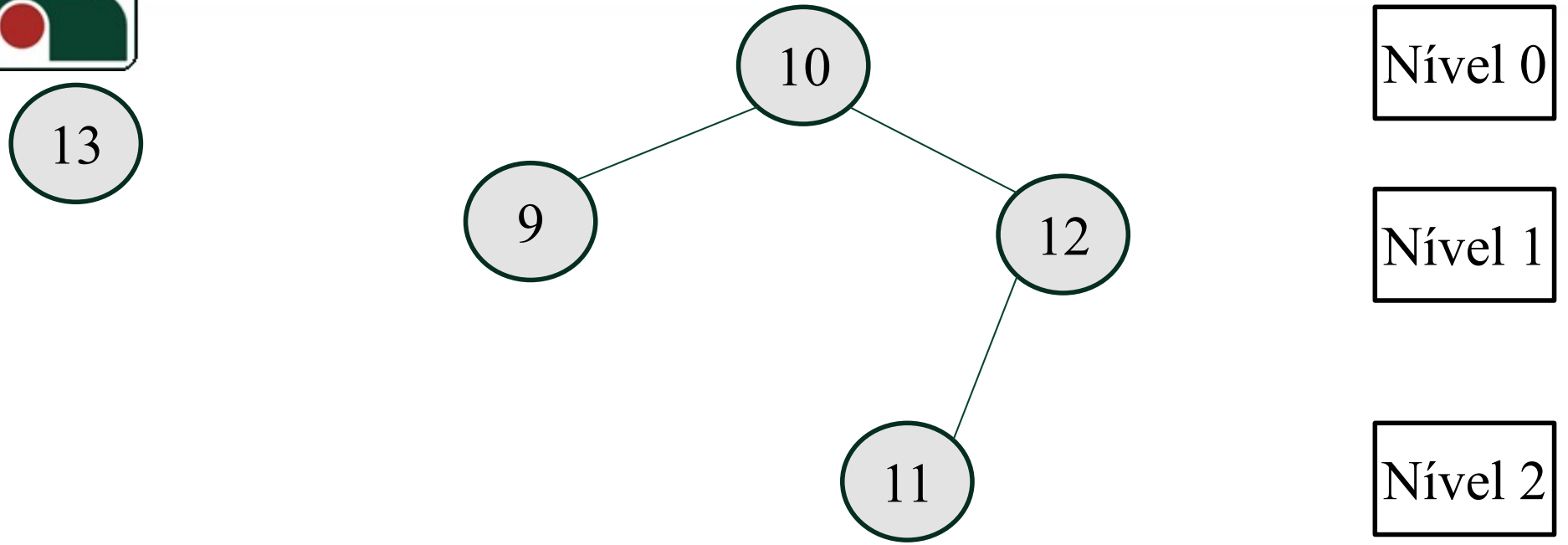
ADT – Árvores Binárias – Aplicação



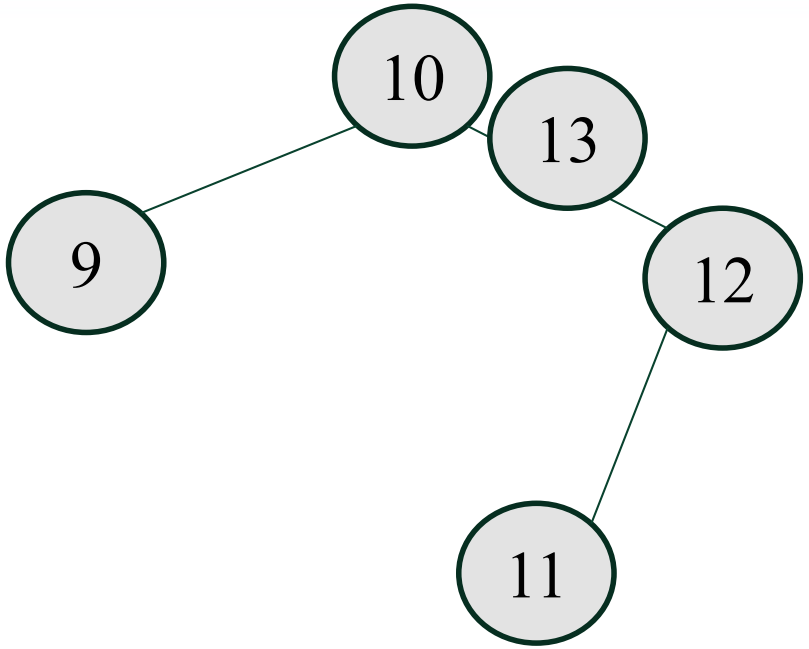
ADT – Árvores Binárias – Aplicação



ADT – Árvores Binárias – Aplicação



ADT – Árvores Binárias – Aplicação

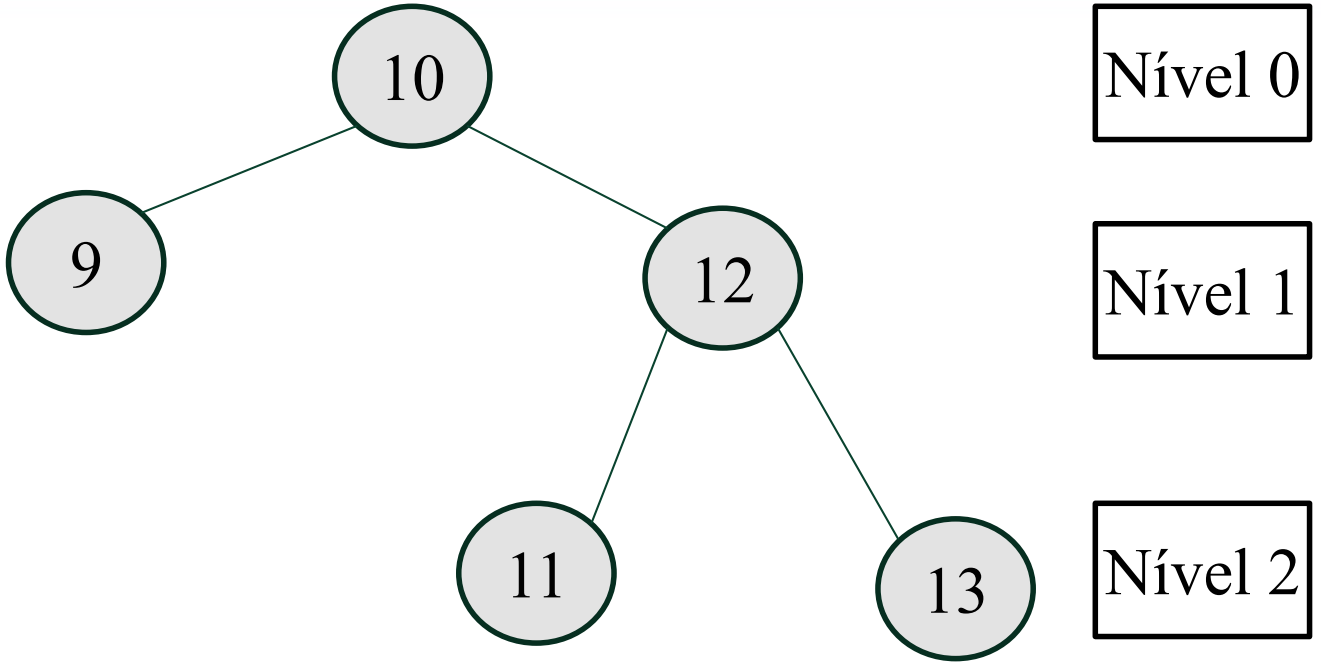


Nível 0

Nível 1

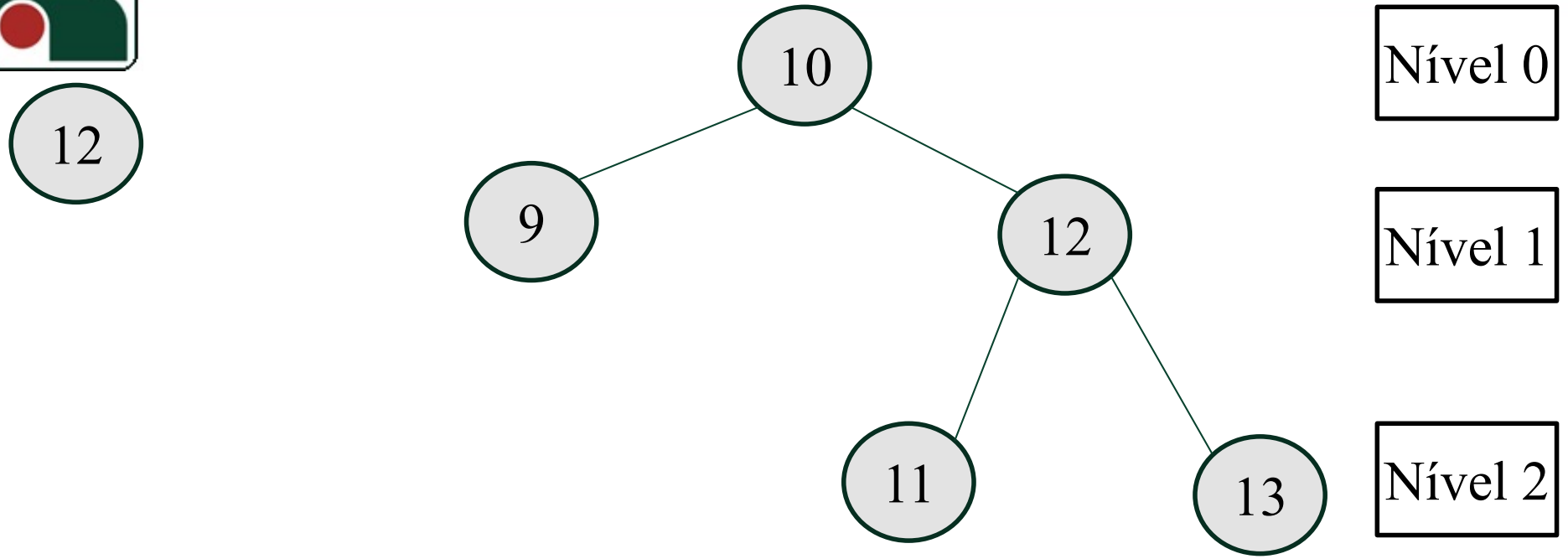
Nível 2

ADT – Árvores Binárias – Aplicação

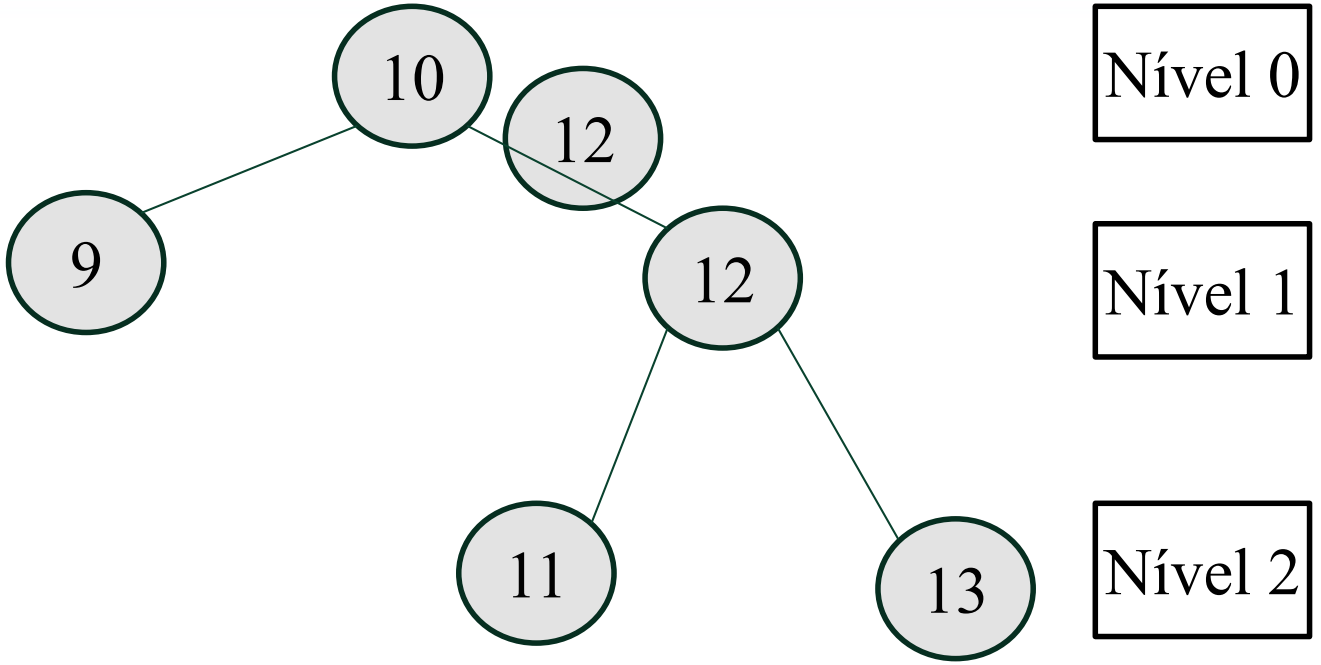


Árvore estritamente binária

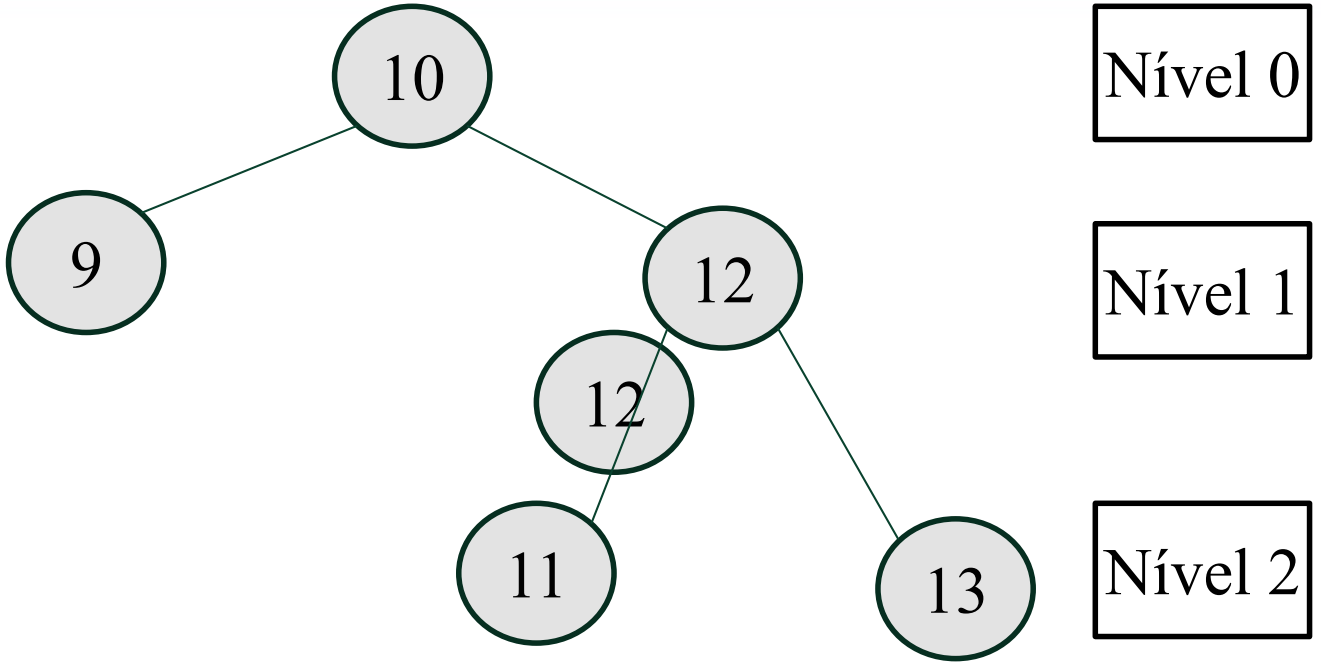
ADT – Árvores Binárias – Aplicação



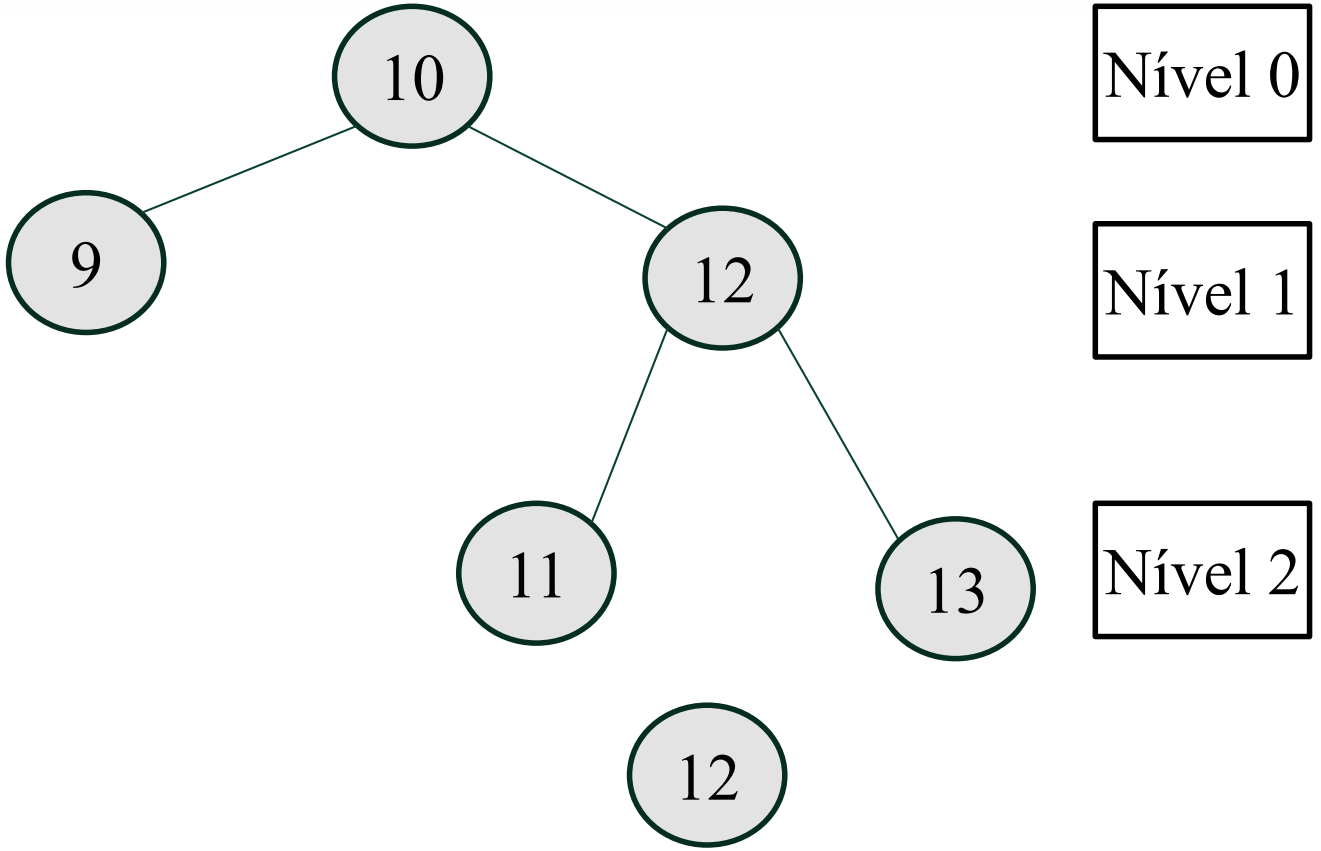
ADT – Árvores Binárias – Aplicação



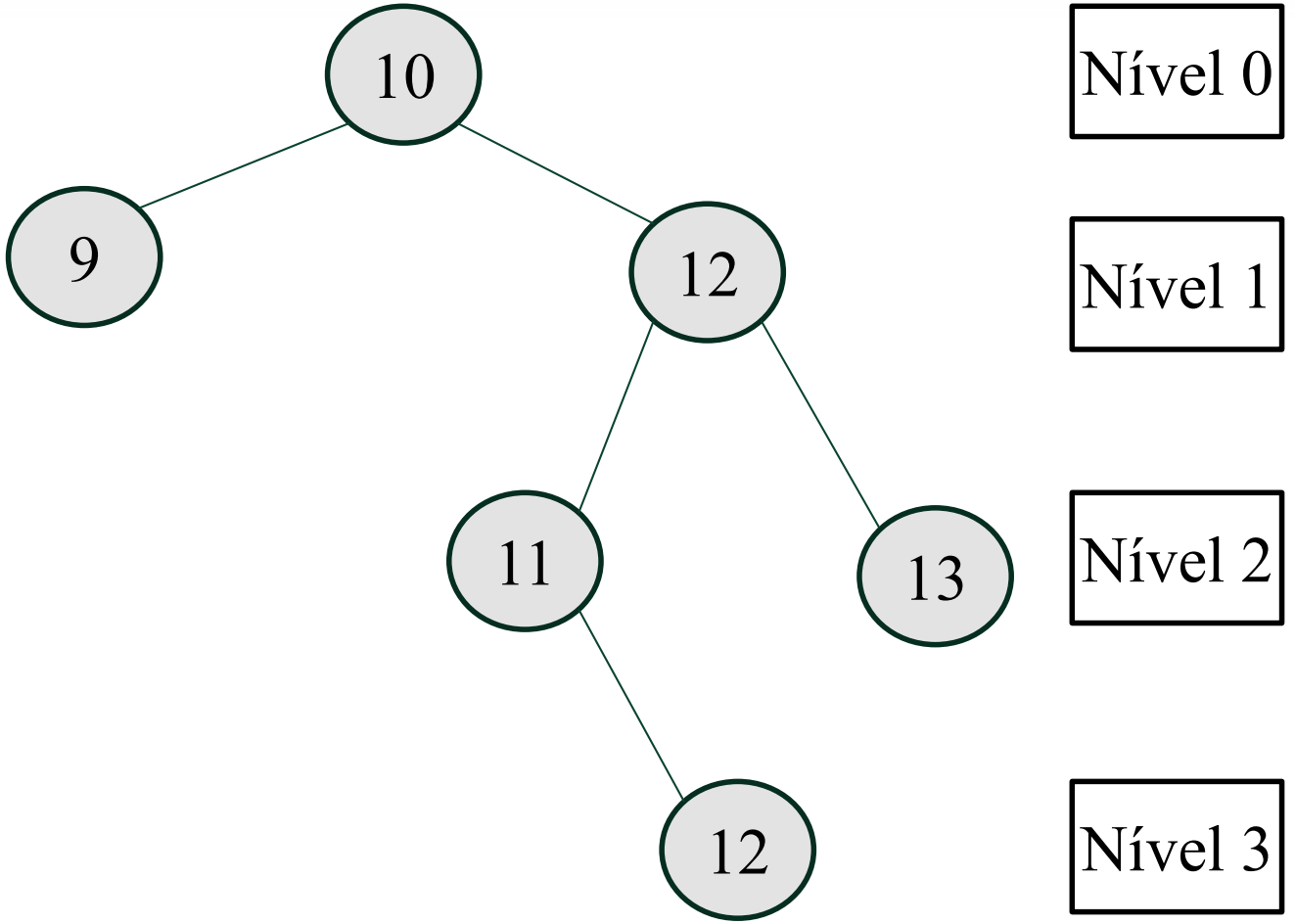
ADT – Árvores Binárias – Aplicação



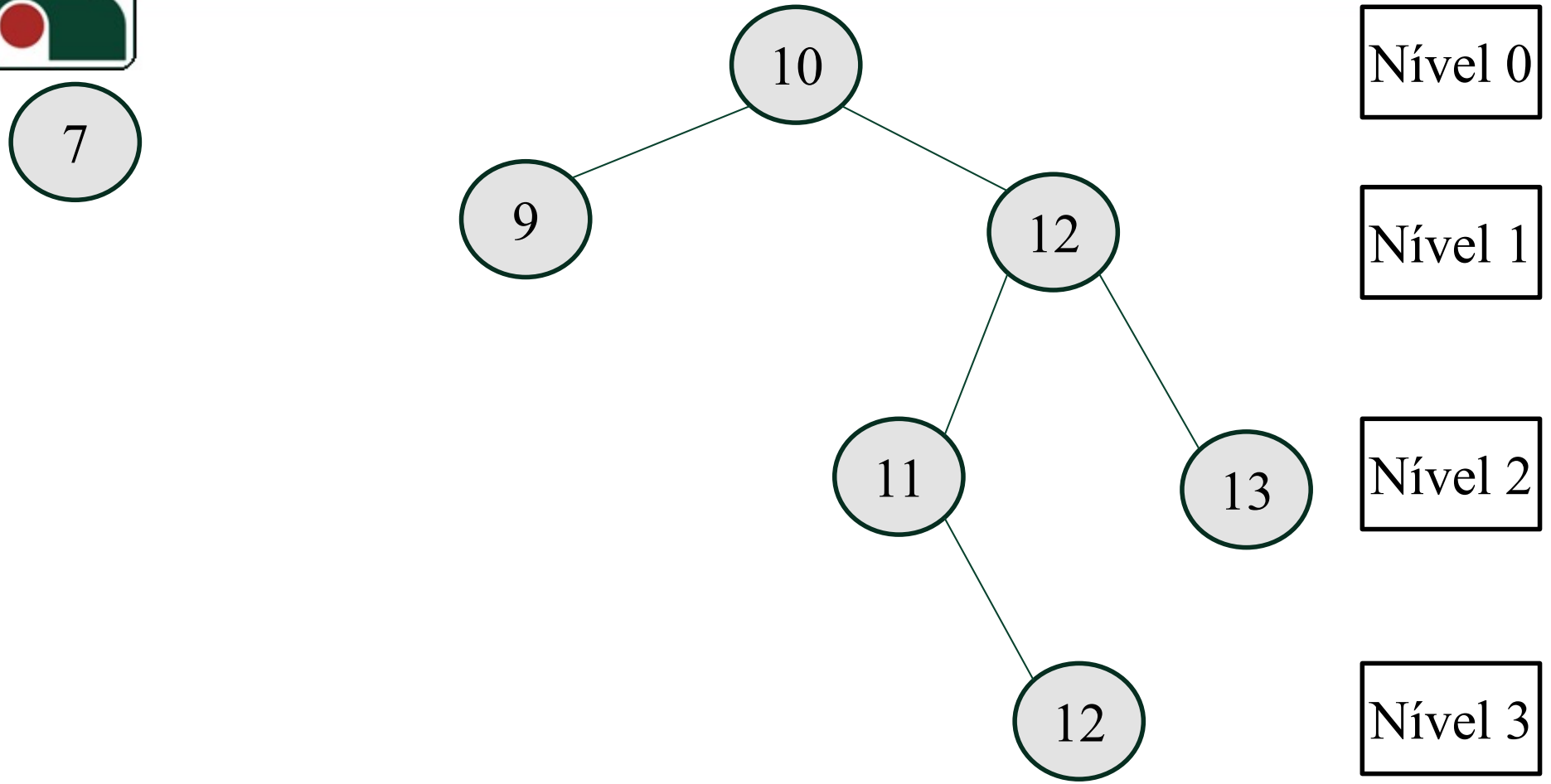
ADT – Árvores Binárias – Aplicação



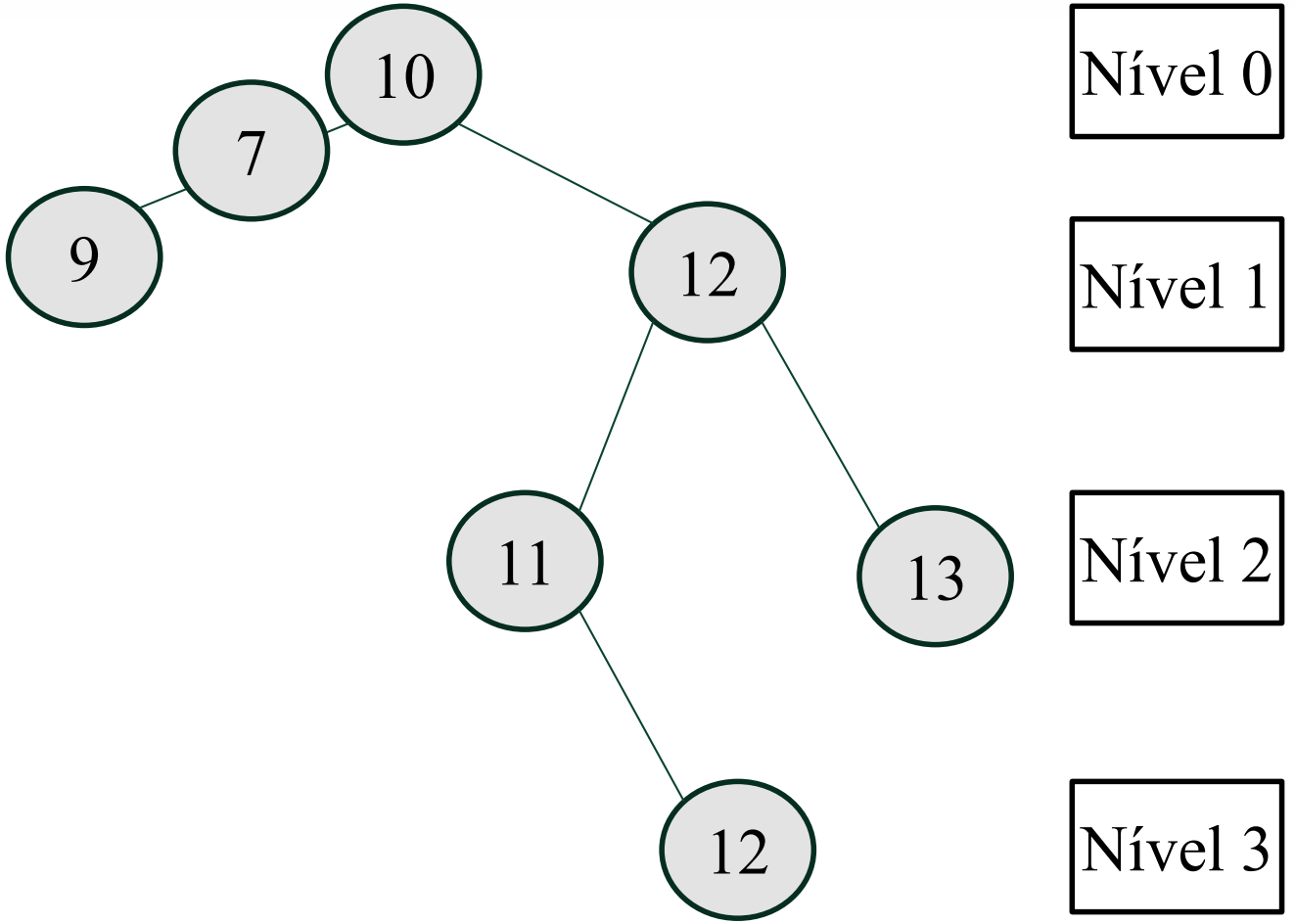
ADT – Árvores Binárias – Aplicação



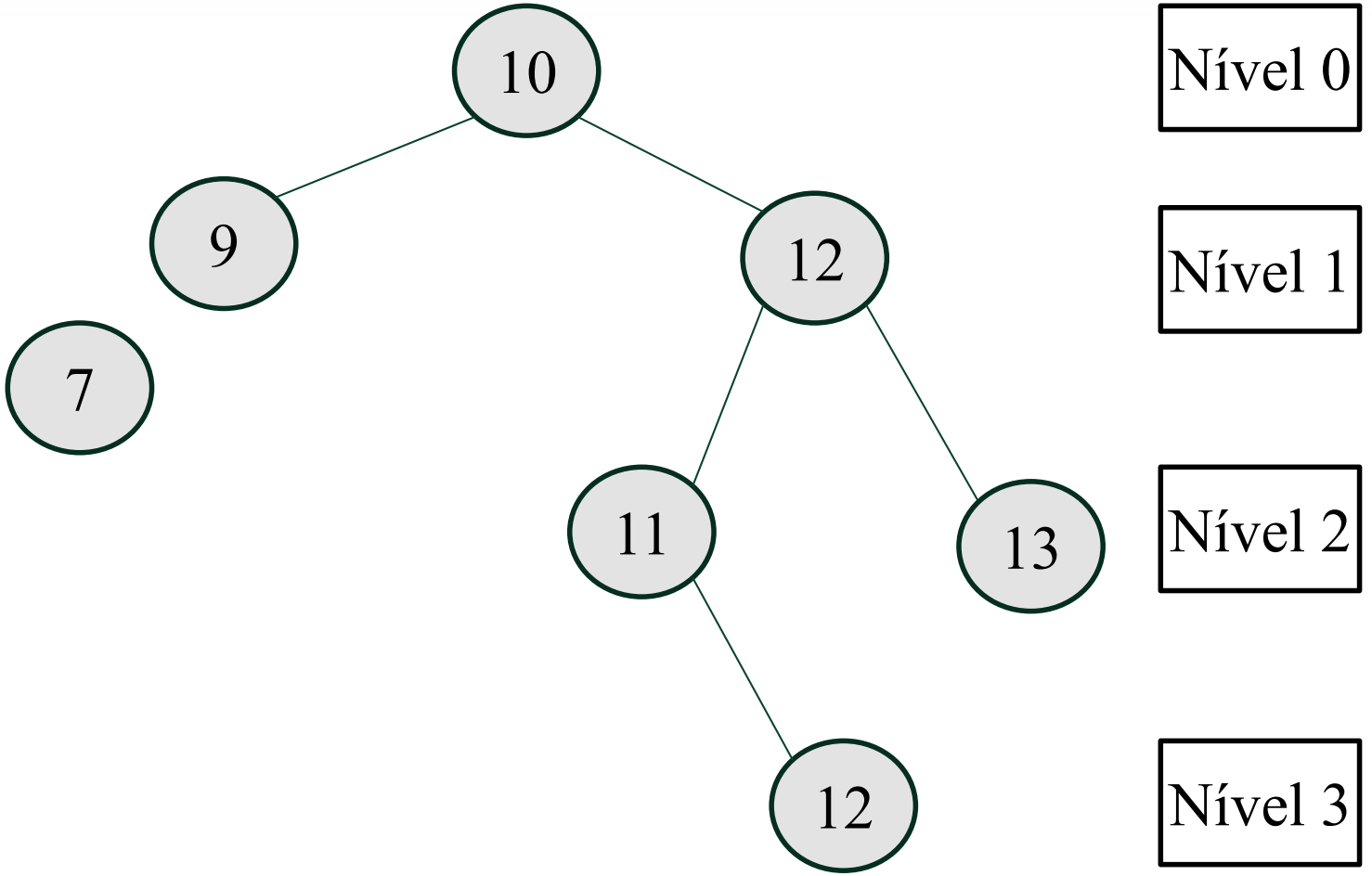
ADT – Árvores Binárias – Aplicação



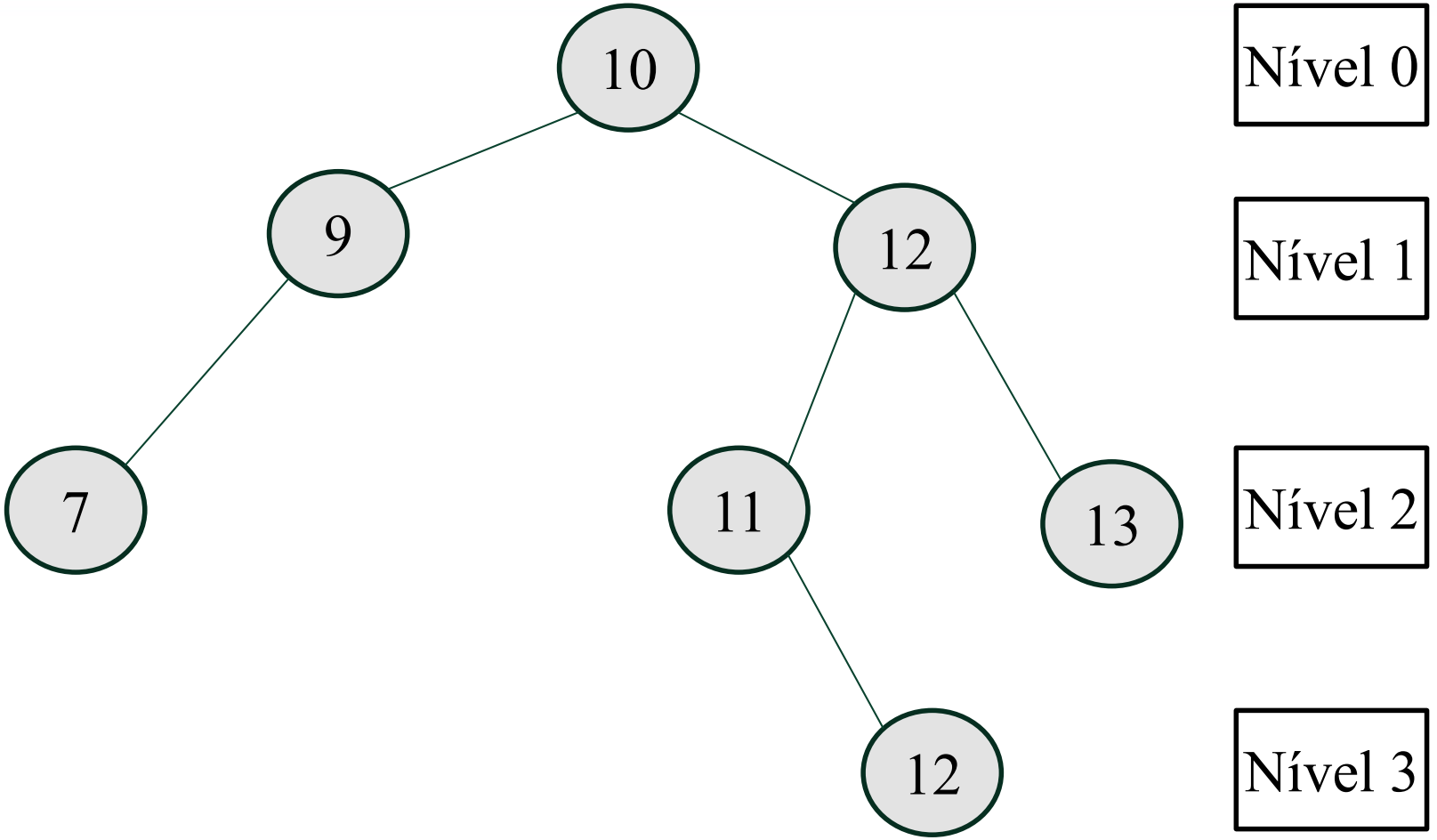
ADT – Árvores Binárias – Aplicação



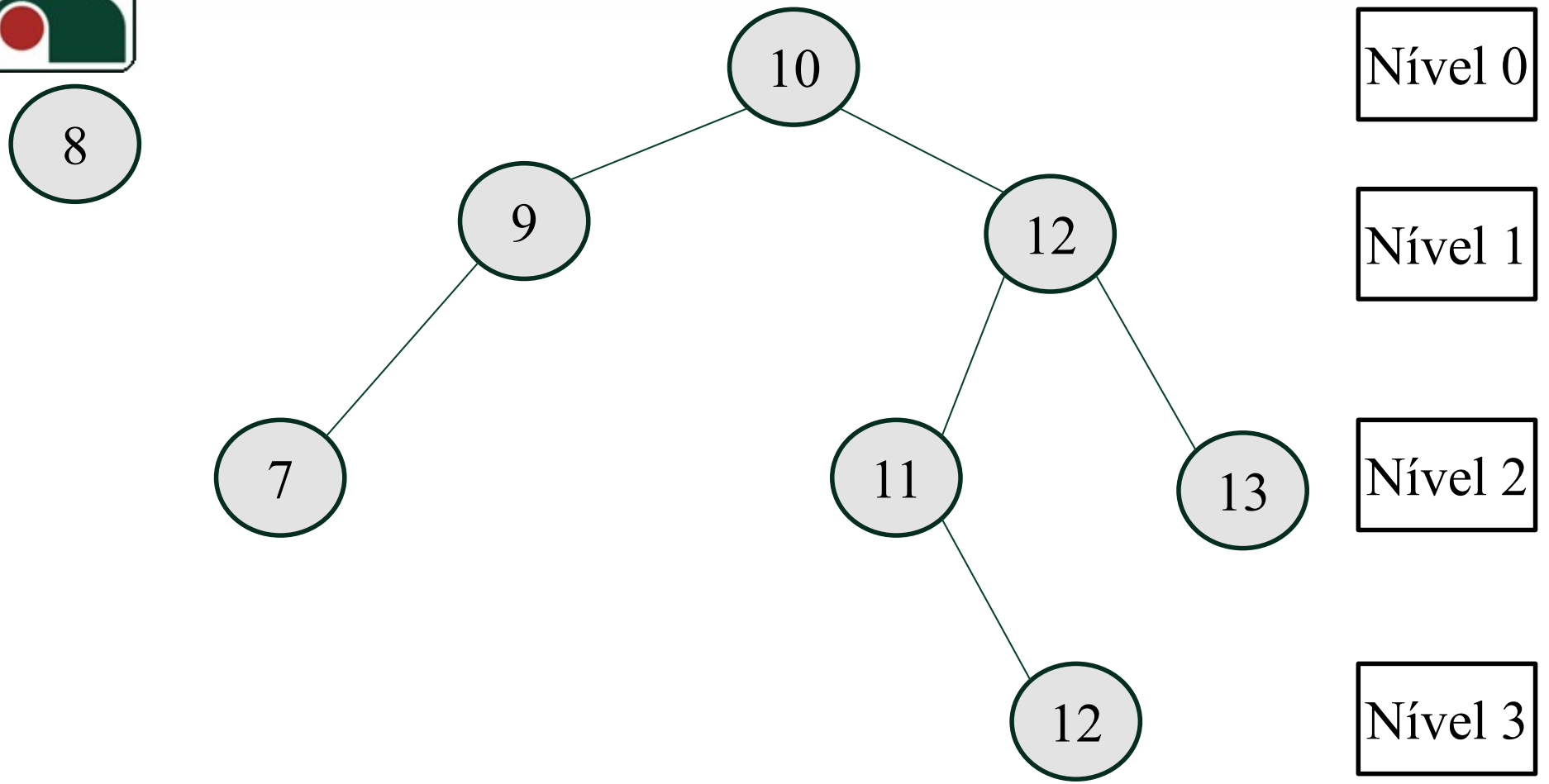
ADT – Árvores Binárias – Aplicação



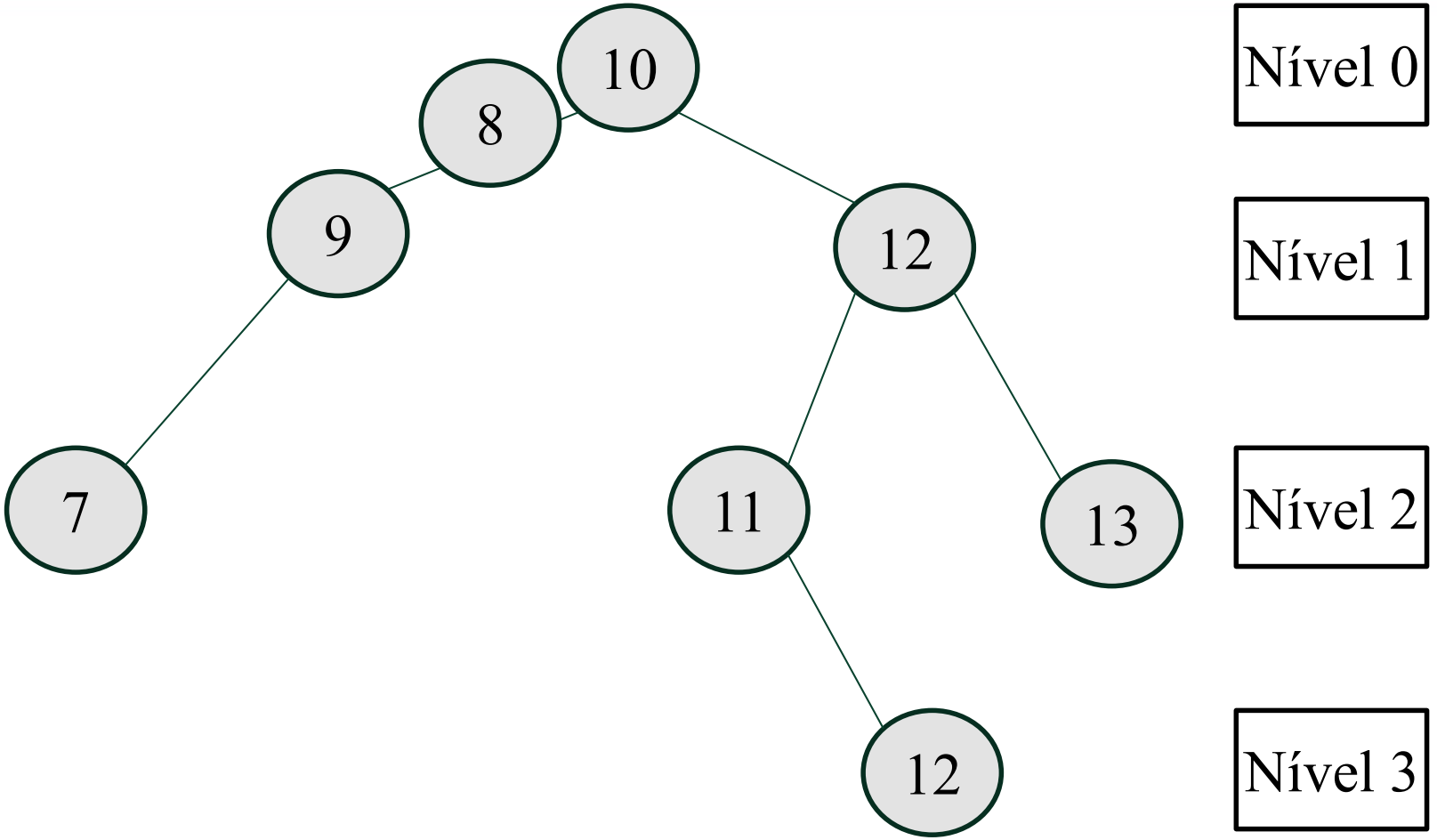
ADT – Árvores Binárias – Aplicação



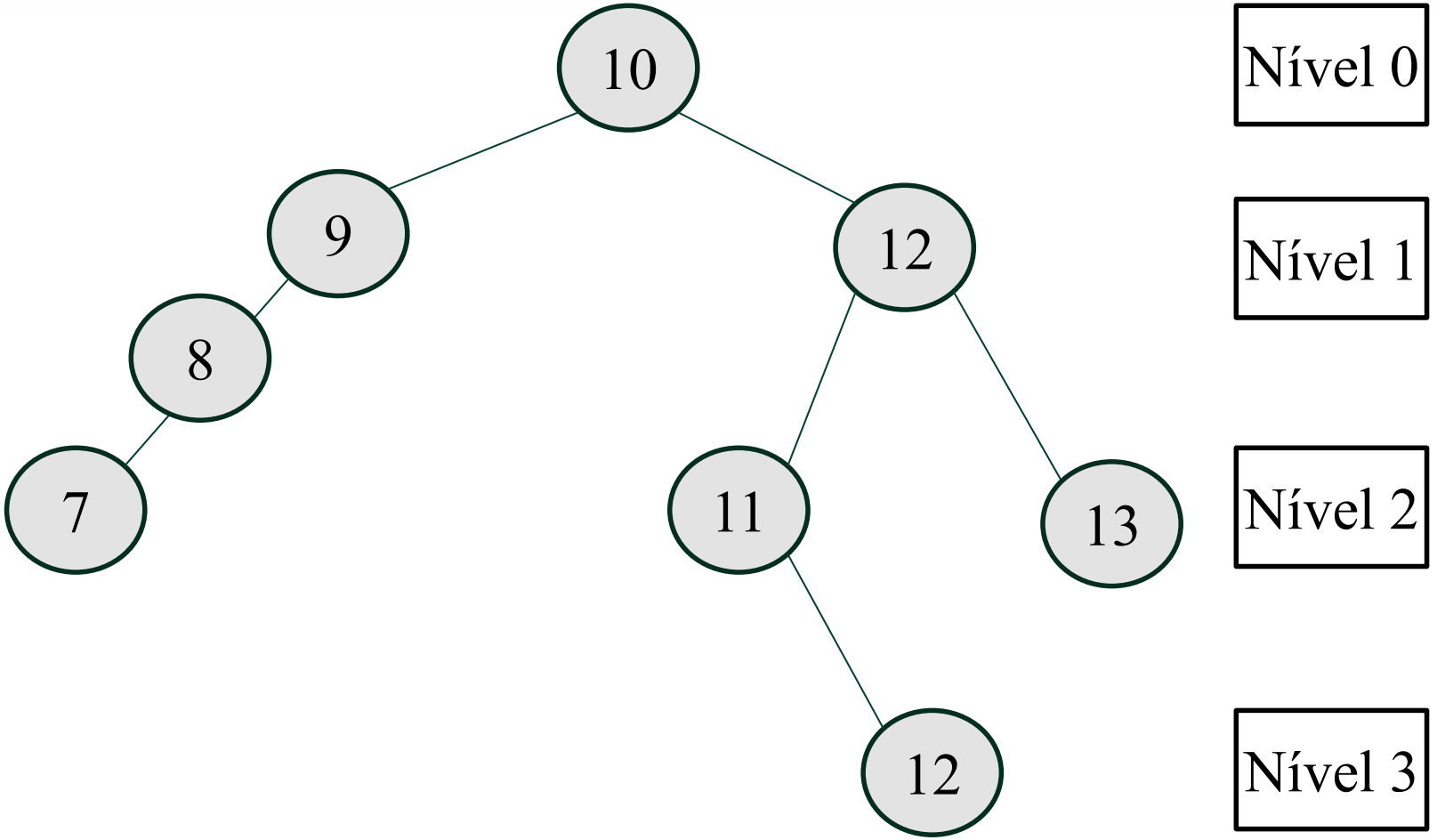
ADT – Árvores Binárias – Aplicação



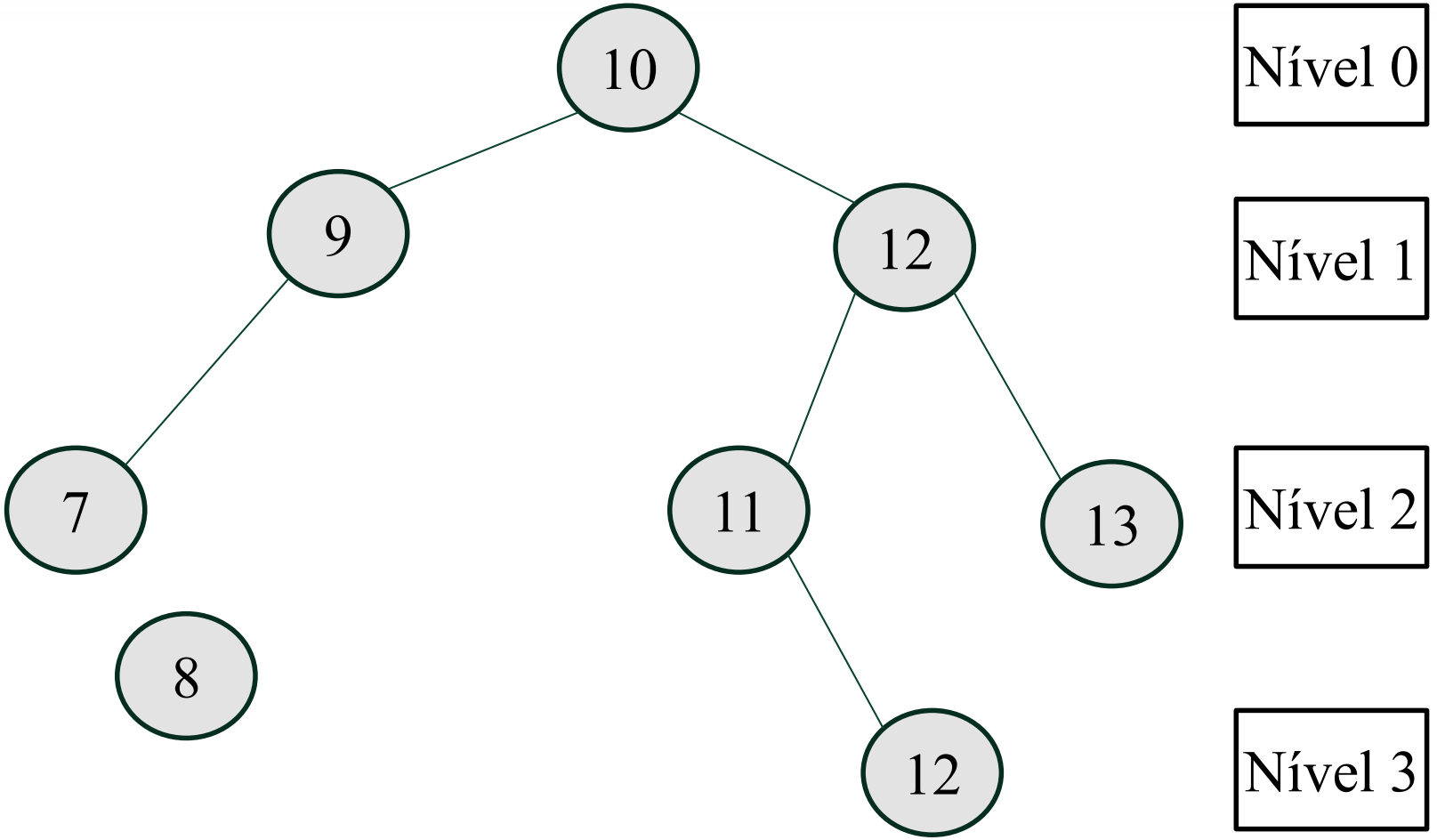
ADT – Árvores Binárias – Aplicação



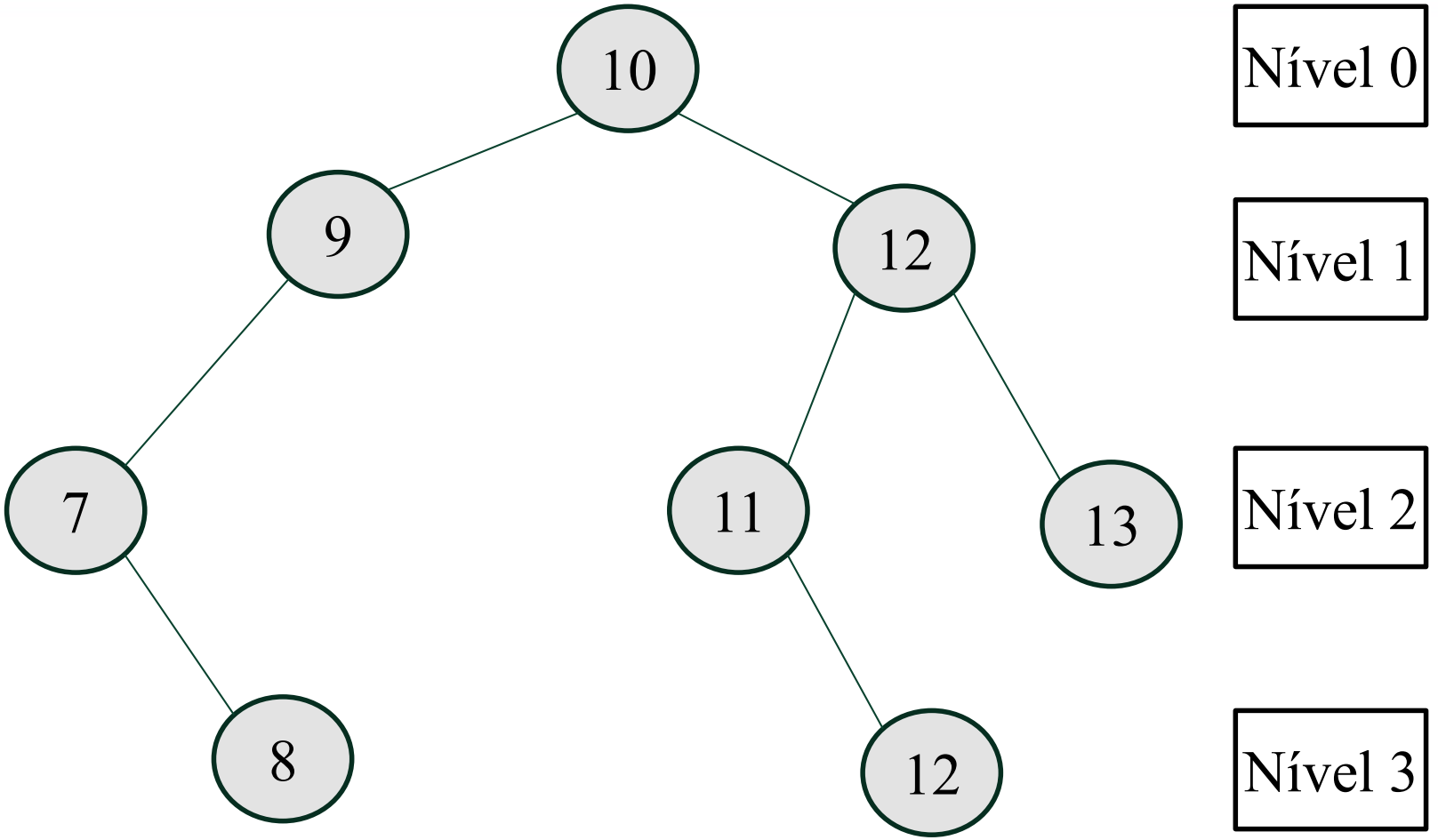
ADT – Árvores Binárias – Aplicação



ADT – Árvores Binárias – Aplicação



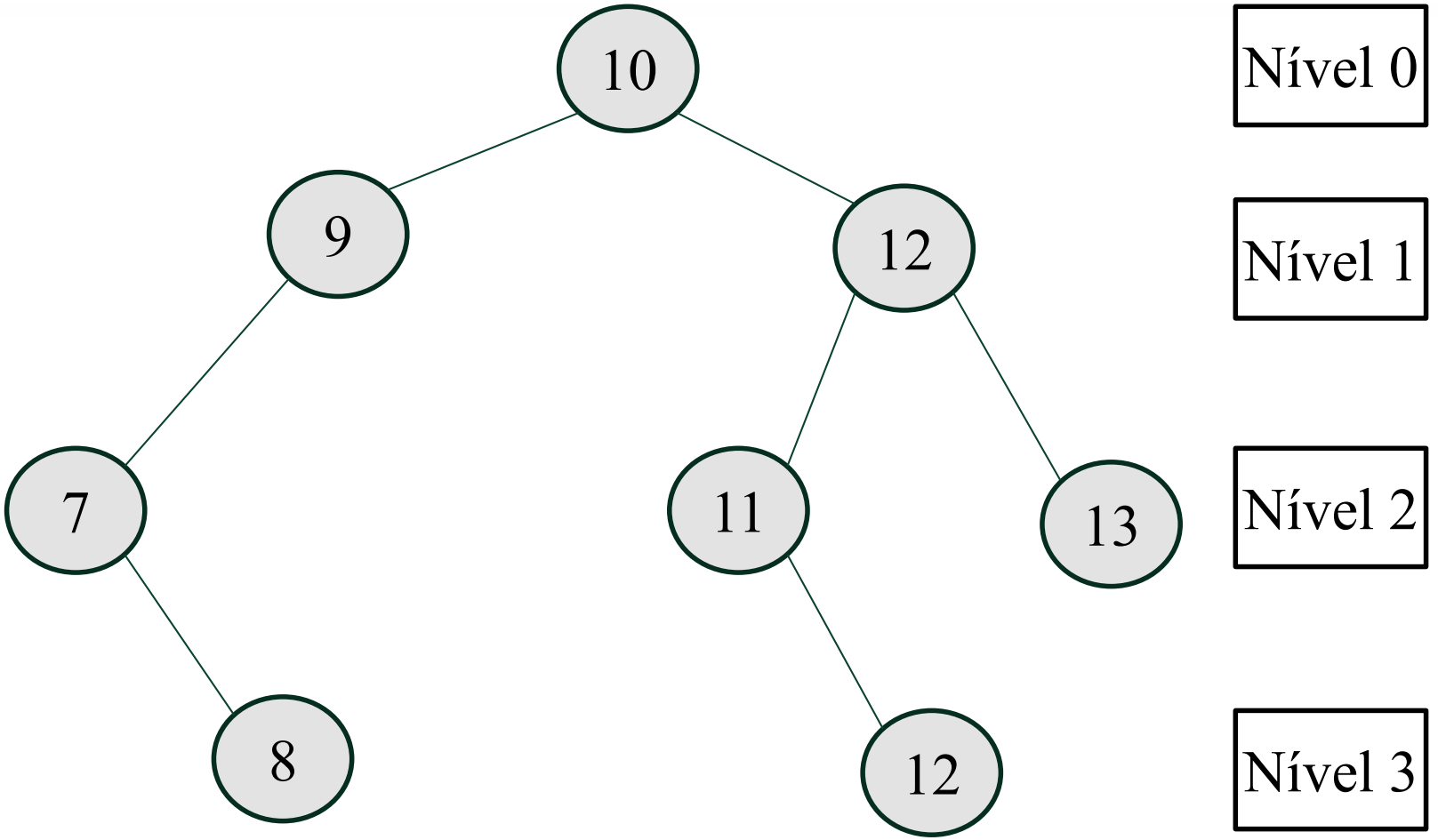
ADT – Árvores Binárias – Aplicação



Árvore não é estritamente binária

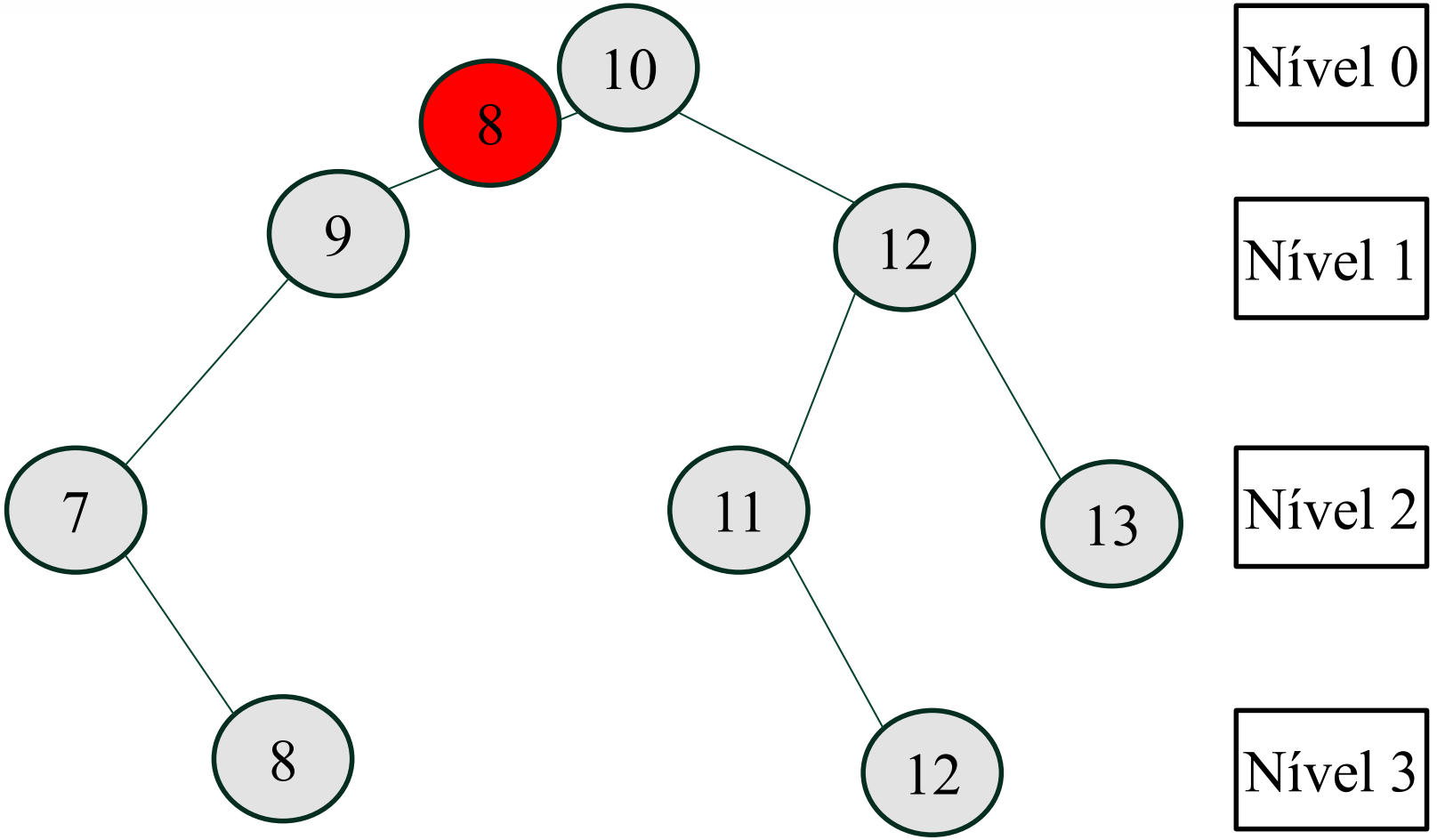
ADT – Árvores Binárias – Aplicação

8

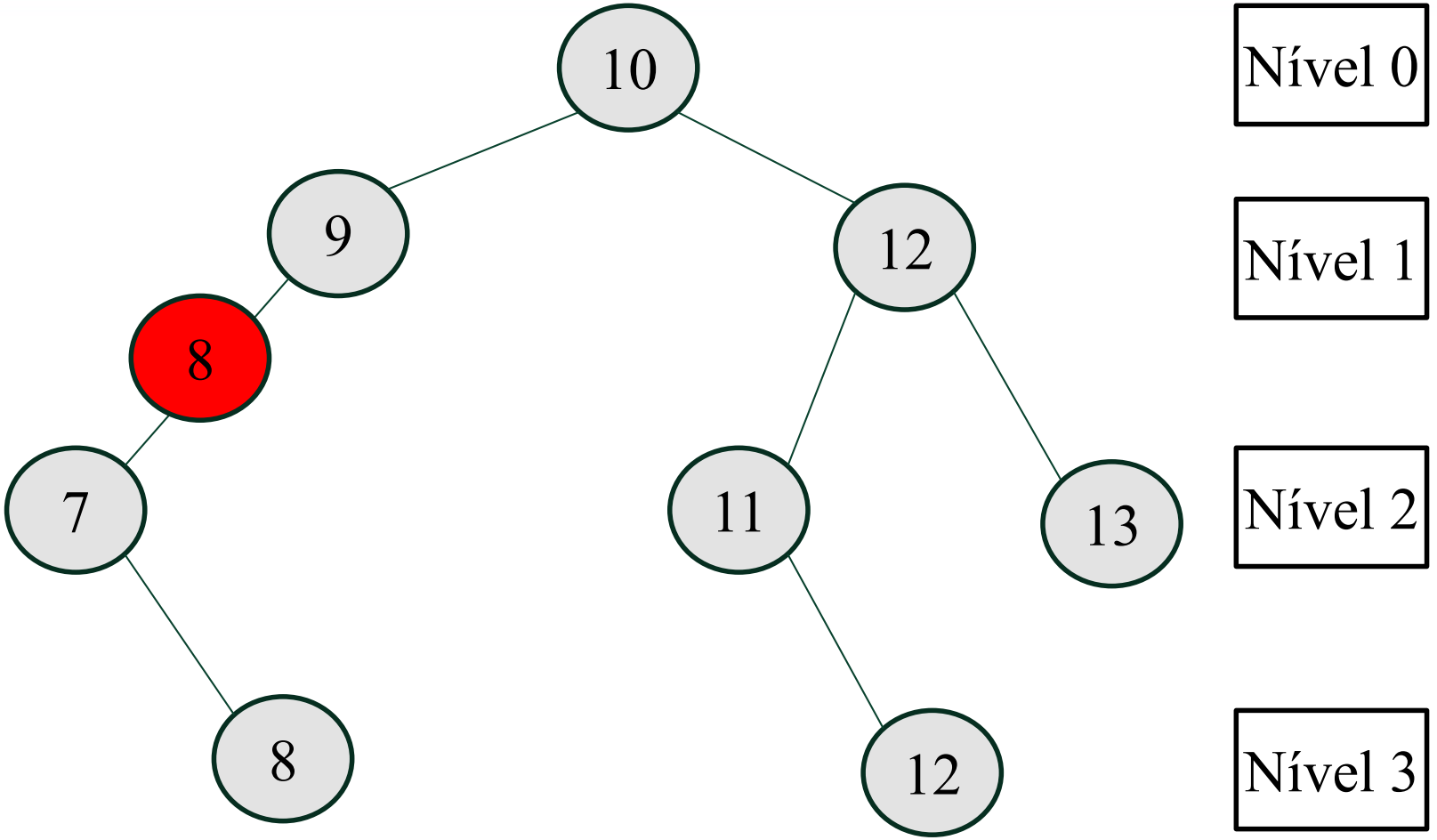


Removendo um valor da árvore

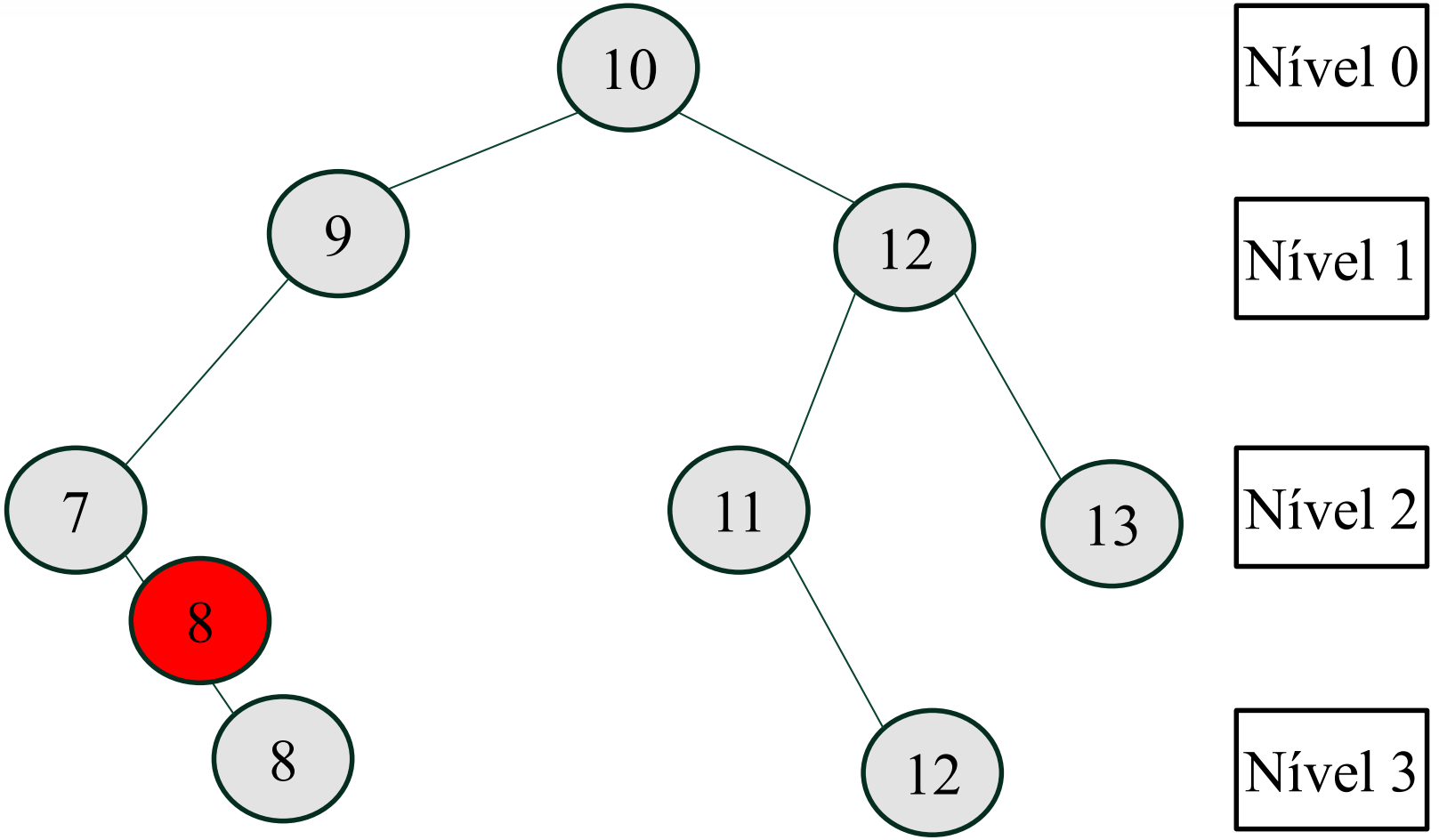
ADT – Árvores Binárias – Aplicação



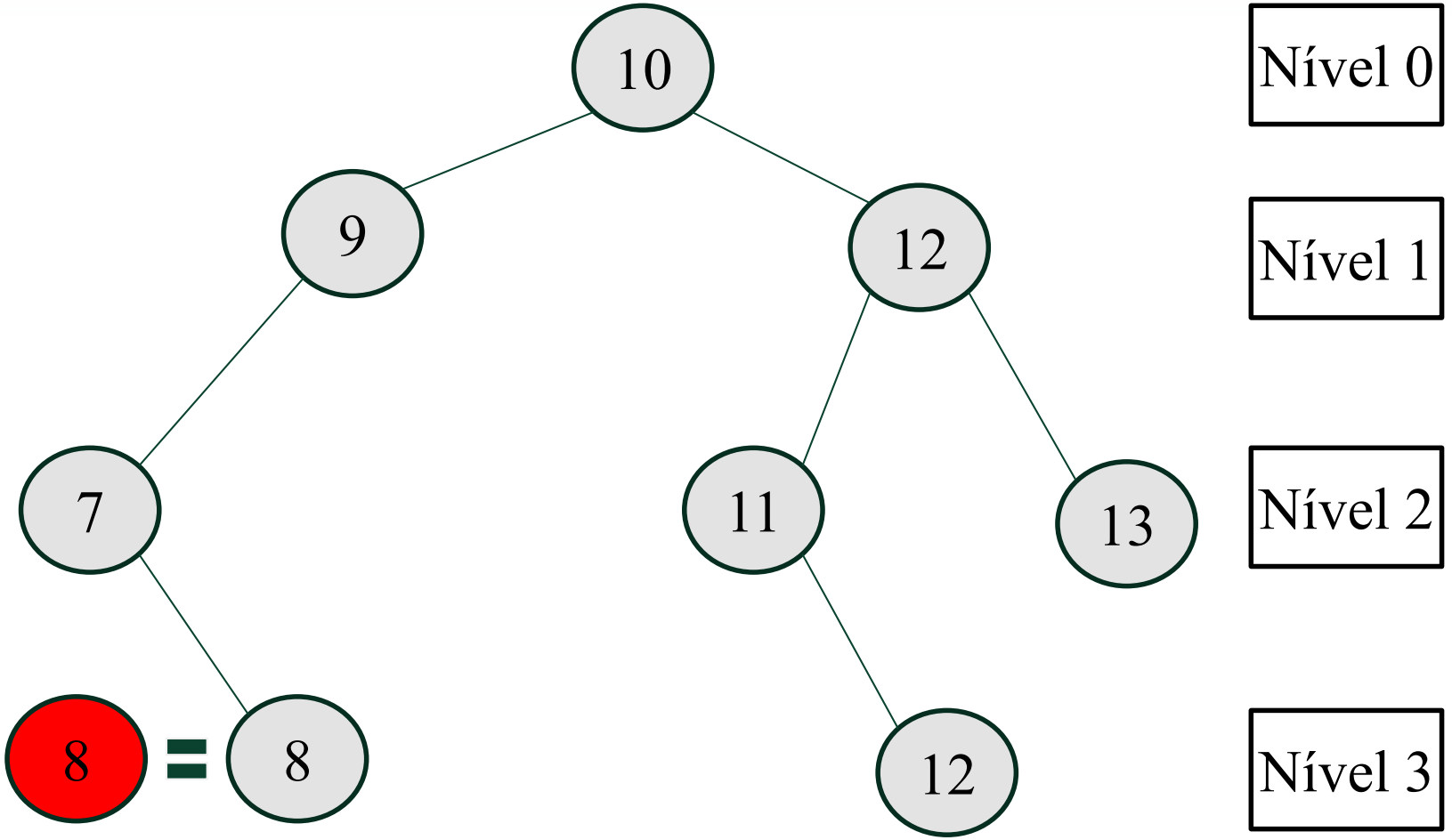
ADT – Árvores Binárias – Aplicação



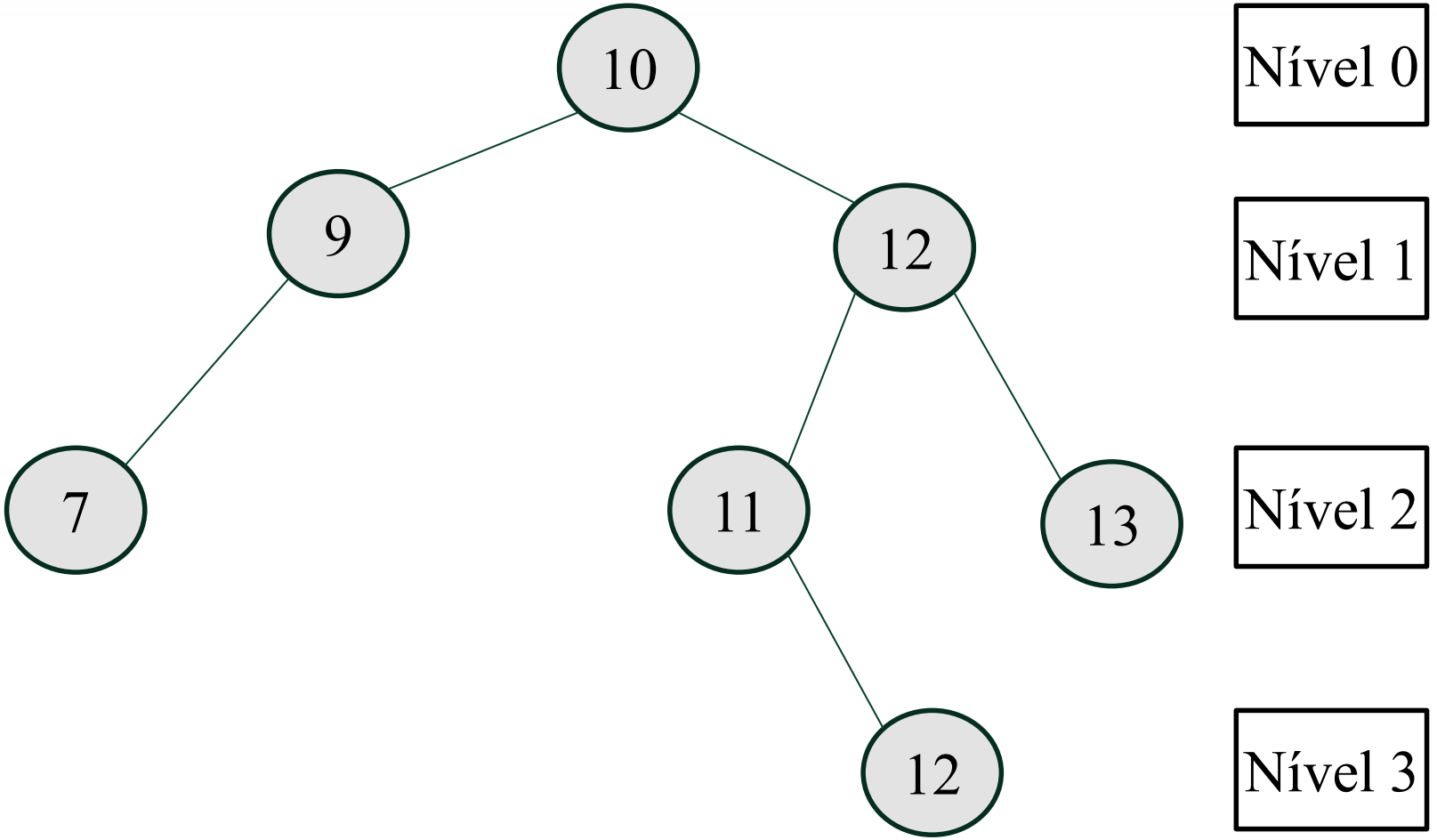
ADT – Árvores Binárias – Aplicação



ADT – Árvores Binárias – Aplicação



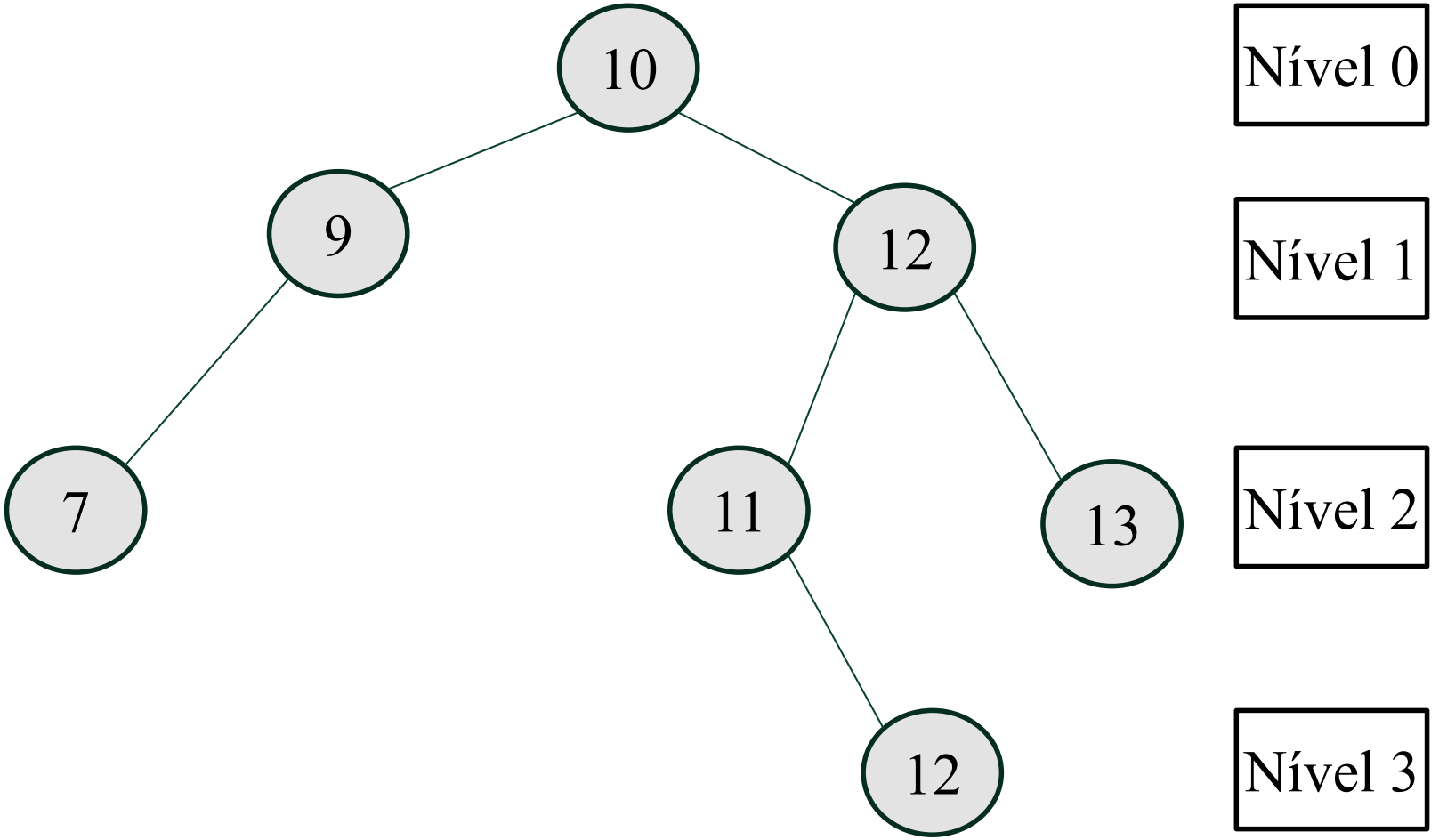
ADT – Árvores Binárias – Aplicação



Valor removido

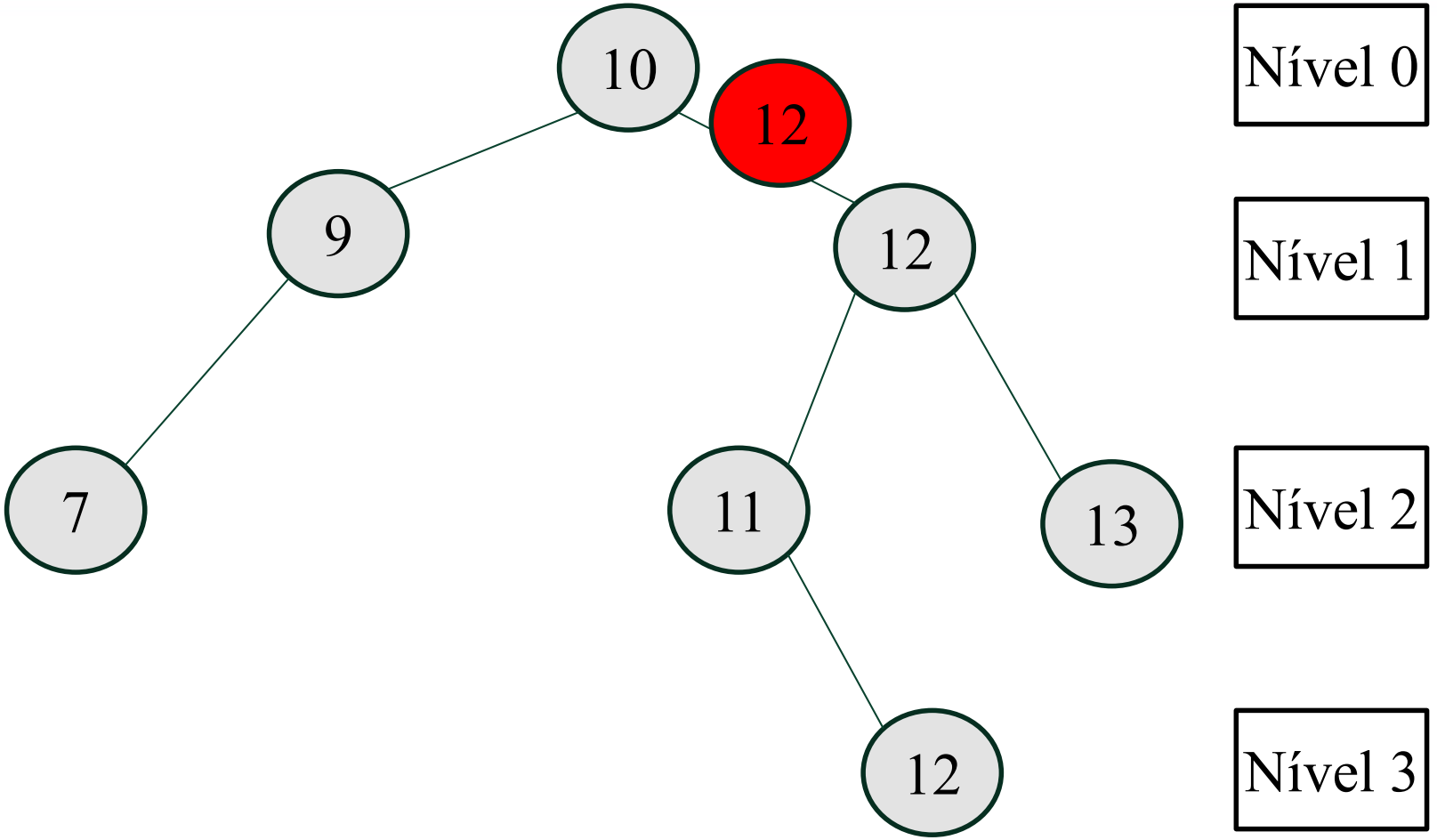
ADT – Árvores Binárias – Aplicação

12

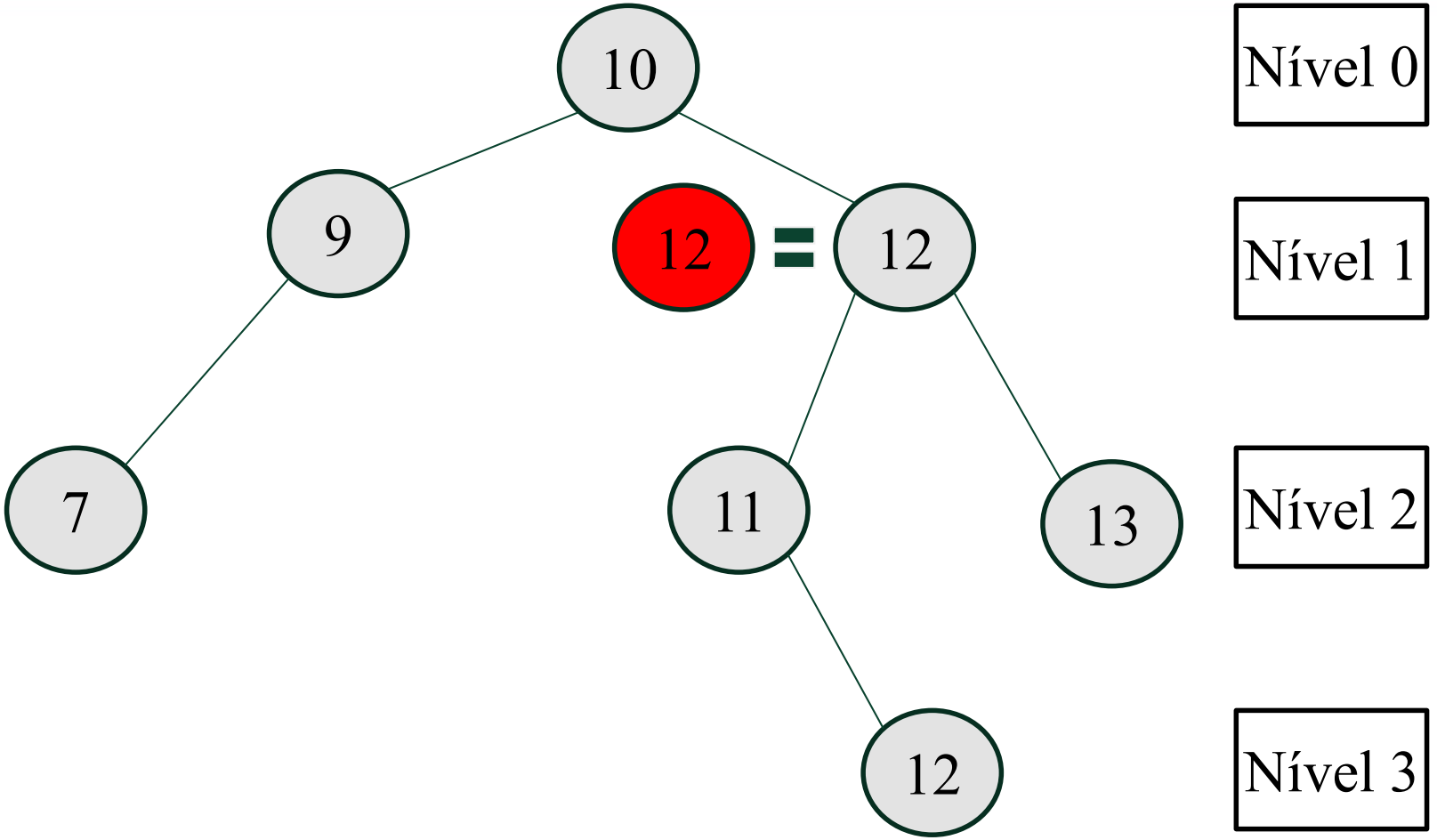


Removendo outro valor da árvore

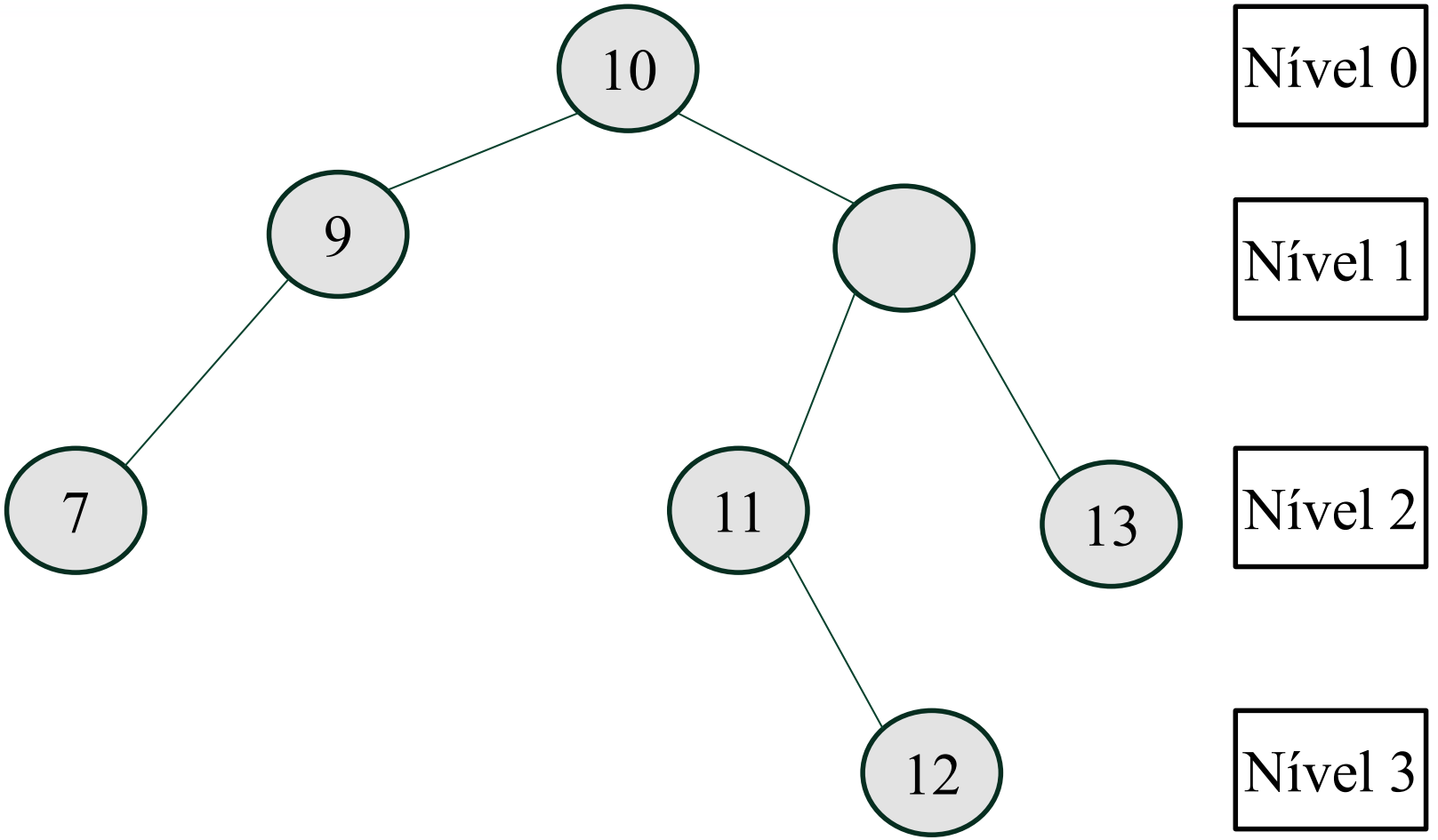
ADT – Árvores Binárias – Aplicação



ADT – Árvores Binárias – Aplicação



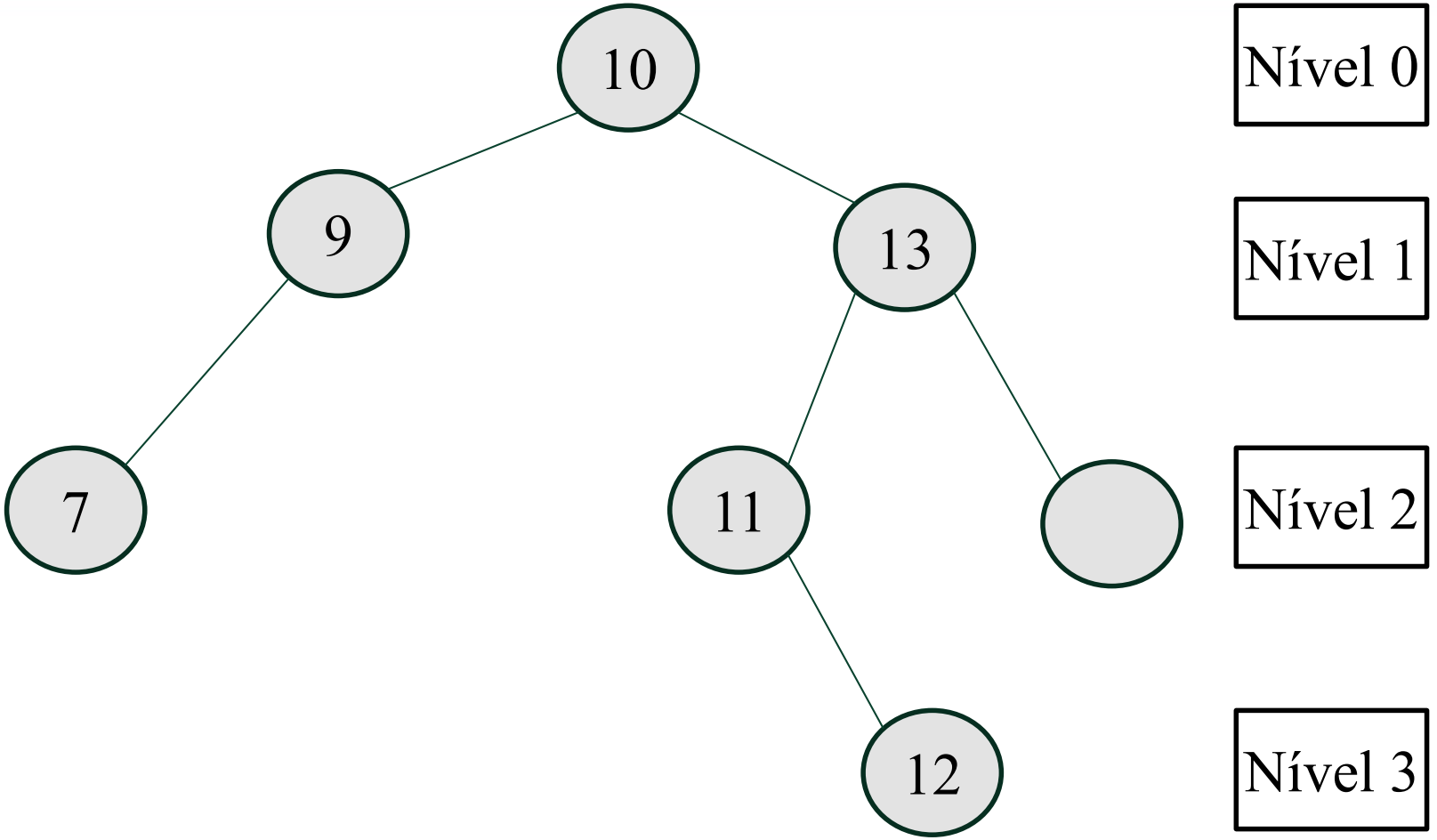
ADT – Árvores Binárias – Aplicação



Valor removido

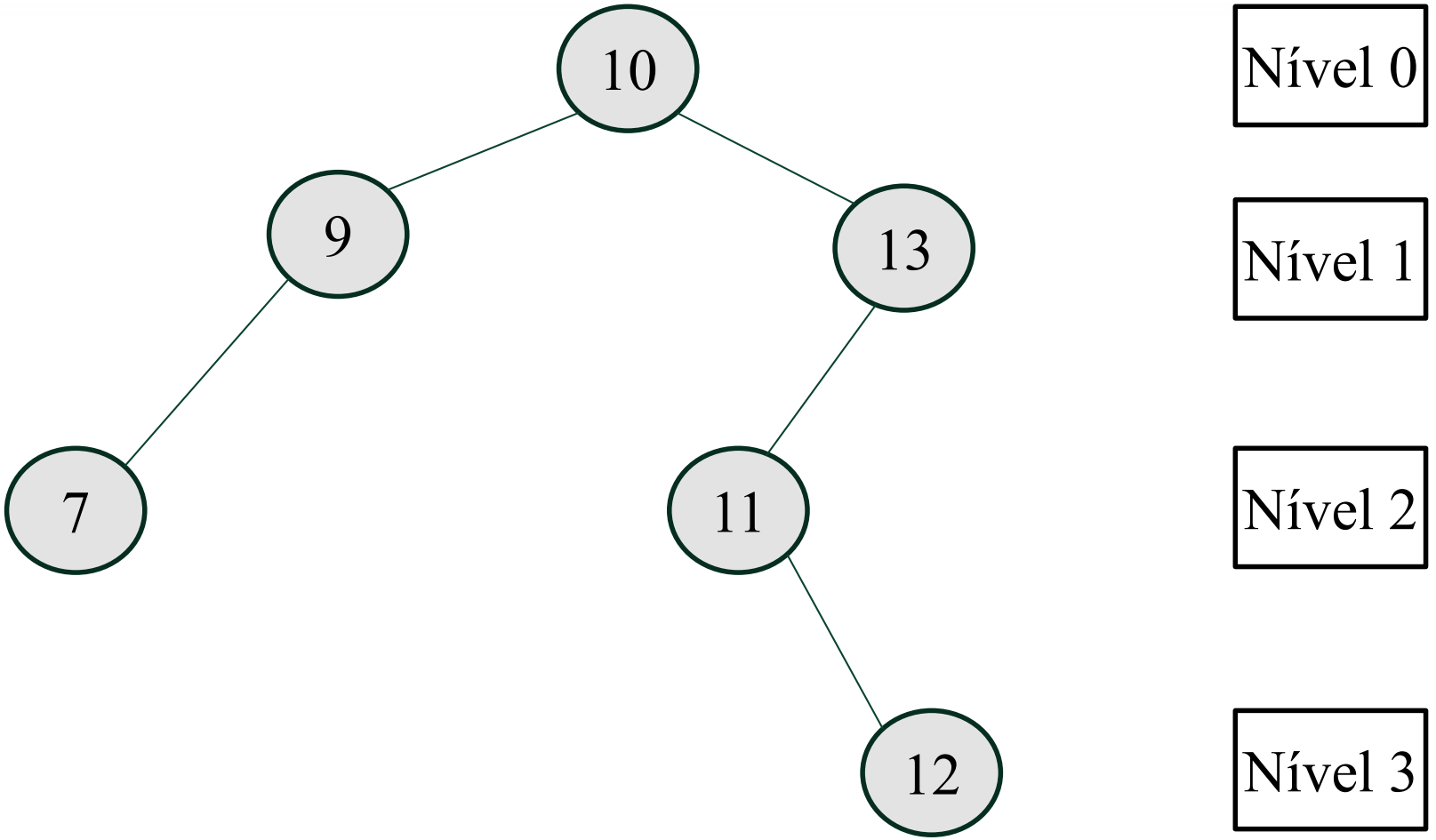
Nessa operação, o filho da direita, que é o mais velho, assume o lugar do nó pai.

ADT – Árvores Binárias – Aplicação



Realocando valores

ADT – Árvores Binárias – Aplicação



Realocando valores

ADT – Árvores Binárias – *Percurso*

Outra operação comum é *percorrer* uma árvore binária, ou seja, percorrer a árvore enumerando cada um de seus nós uma vez.

Pode-se simplesmente querer imprimir o conteúdo de cada nó ao enumerá-lo, ou se pode processá-lo de alguma maneira.

Seja qual for o caso, se fala em *visitar* cada nó à medida que ele é enumerado.

ADT – Árvores Binárias – *Percurso*

A ordem na qual os nós de uma lista linear são visitados num percurso é do primeiro para o último (ou vice-versa em listas duplas).

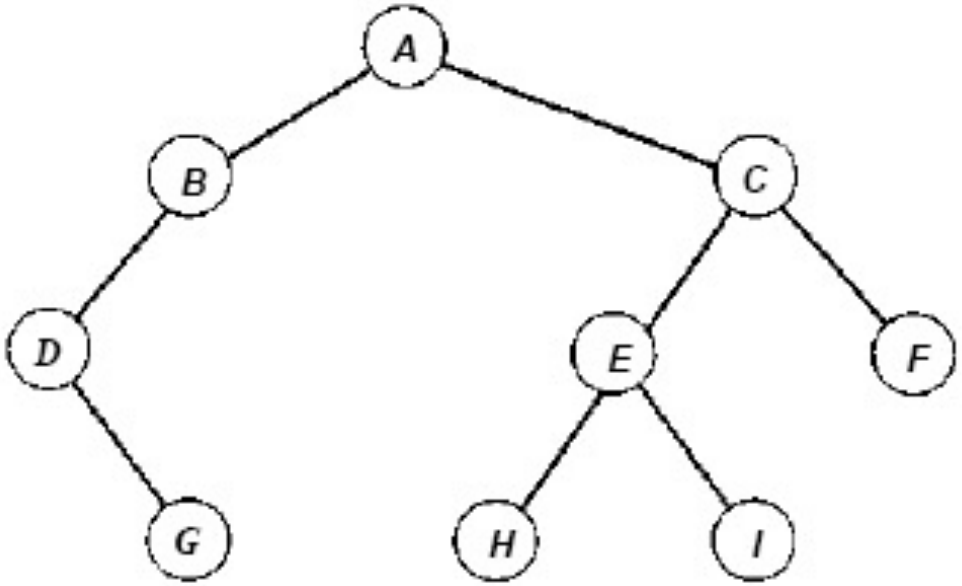
Não existe uma ordem "natural" para os nós de uma árvore.
Sendo assim, são usados diferentes ordenamentos de percurso em diferentes casos.

Definir-se-á três desses métodos de percurso. Em cada um desses métodos, não é preciso fazer nada para percorrer uma árvore binária vazia. Todos os métodos podem ser definidos recursivamente, de modo que **percorrer uma árvore binária envolve visitar a raiz e percorrer suas subárvores esquerda e direita**. A única diferença entre os métodos é a ordem na qual essas três operações são efetuadas.

ADT – Árvores Binárias – *Percurso*

Para percorrer uma árvore binária não-vazia em ***pré-ordem*** (conhecida também como *percurso em profundidade*), efetuam-se as três seguintes operações:

1. Visita-se a **raiz**.
2. Percorre a subárvore **esquerda** em ordem prévia (raiz e filhos – esq e dir).
3. Percorre a subárvore **direita** em ordem prévia.

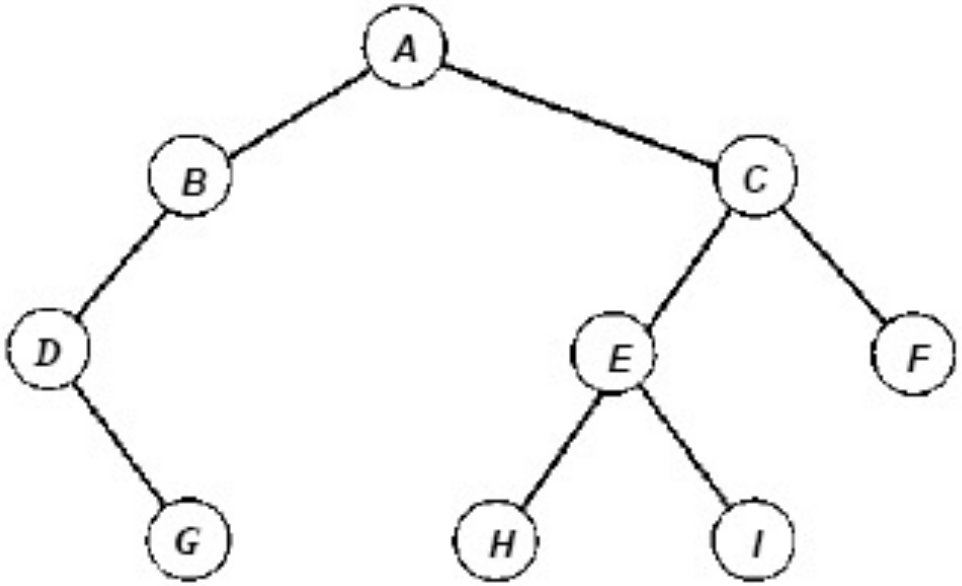


Pré-ordem: *ABDGCEHIF*

ADT – Árvores Binárias – *Percurso*

Para percorrer uma árvore binária não-vazia em ***em ordem*** (ou ordem simétrica – regular, igual):

1. Percorre a subárvore **esquerda** em ordem simétrica (esq – raiz – dir).
2. Visita-se a **raiz**.
3. Percorre a subárvore **direita** em ordem simétrica.

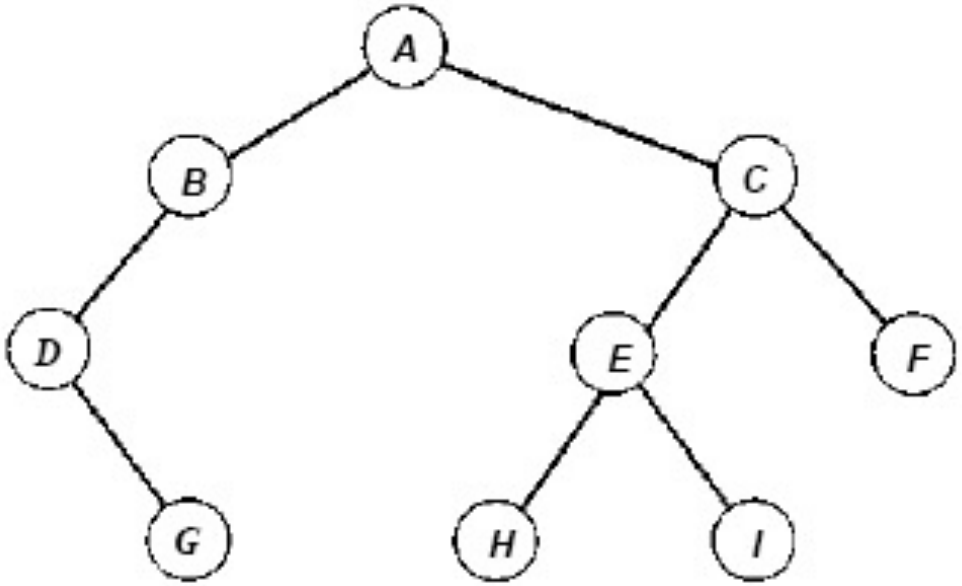


Em ordem: *DGBAHEICF*

ADT – Árvores Binárias – *Percurso*

Para atravessar uma árvore binária não-vazia em ***pós-ordem***:

1. Percorre a subárvore **esquerda** em ordem posterior (esq – dir – raiz).
2. Percorre a subárvore **direita** em ordem posterior.
3. Visita-se a **raiz**.



Pós-ordem: *GDBHIEFCA*

ADT – Árvores Binárias – Por quê?

Por que usar árvores binárias?

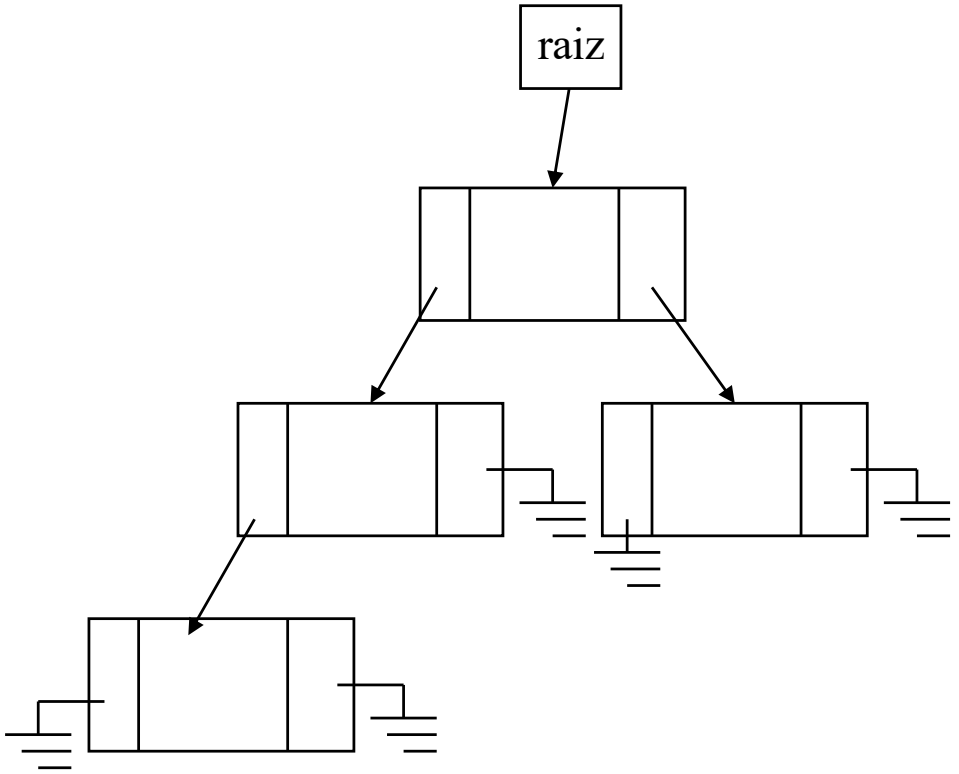
Geralmente, porque ela combina as vantagens de duas outras estruturas: um vetor ordenado e uma lista encadeada.

Pode-se buscar numa árvore rapidamente, como em um vetor ordenado, e pode-se também inserir e eliminar itens rapidamente, como em uma lista encadeada.

ADT – Árvores Binárias – Implementação

Representação de nós em árvores binárias:

Como acontece com os nós de listas, os nós de árvores podem ser implementados como elementos de vetores ou como alocações de uma variável dinâmica.



ADT – Árvores Binárias – *Maketree*

Vamos conhecer o código de implementação em Java para inserir nós numa árvore?

ADT – Árvores Binárias – *Definindo Main*

Em Java:

```
class J_Exemplo {     // NÃO RECURSIVO

    public static void main(String args[]){

       Arvore arv = new Arvore();
       int n;

       do {
          n = ...("Digite o Elemento(tecle 0 para parar): ");
          if(n != 0)
             arv.cria_arvore(n);
       } while (n != 0);

    }
```

ADT – Árvores Binárias – *Definindo classes*

```
class Node {
    int info;
    Node dir;
    Node esq;
}

class Arvore {
    Node raiz = null;
    public void cria_arvore(int n){...}
}
```

ADT – Árvores Binárias – *Criar árvore*

```
public void cria_arvore(int n){     // NÃO RECURSIVO
        Node temp_no = new Node();
        temp_no.info = n;
        if(raiz == null)
             raiz = temp_no;
        else{
             Node atual = raiz;
             Node pai;
             while(true){
                  pai = atual;
                  if (n < atual.info){
                      atual = atual.esq;
                      if (atual == null){
                          pai.esq = temp_no;
                          return;
                      }
                  } else {
                      atual = atual.dir;
                      if (atual == null){
                          pai.dir = temp_no;
                          return;
                      }
                  }
           }}}
```

ADT – Árvores Binárias – *Definindo Main*

```
class J_Exemplo {          // RECURSIVO

    public static void main(String args[]){

        Node raiz = null;
        Arvore arv = new Arvore();
        int n;

        do {
            n = ...("Digite o Elemento(tecle 0 para parar): ");
            if(n != 0)
                raiz = arv.inserir(raiz,n);
        } while (n != 0);

    }
```

ADT – Árvores Binárias – *Criar árvore*

```
public Node inserir(Node aux, int n)      // RECURSIVO
    {
        if(aux == null){
           aux = new Node();
           aux.info = n;
           aux.esq = aux.dir = null;
        }
        else if (n < aux.info)
                aux.esq = inserir(aux.esq,n);
            else
               aux.dir = inserir(aux.dir,n);
        return aux;
    }
```

ADT – Árvores Binárias – *Maketree*

Agora o código de implementação em Python para inserir nós numa árvore...

ADT – Árvores Binárias – *Definindo classes*

```
class Node:

    def __init__(self):
        self.__info = 0
        self.__dir = self.__esq = None
    def getInfo(self):
        return self.__info
    def setInfo(self, info):
        self.__info = info
    def getDir(self):
        return self.__dir
    def setDir(self, dir):
        self.__dir = dir
    def getEsq(self):
        return self.__esq
    def setEsq(self, esq):
        self.__esq = esq
```

ADT – Árvores Binárias – *Criar árvore*

```
class Arvore:

    def __init__(self):
        self.__raiz = None

    def inserir(self, aux, n):  # RECURSIVO
        if aux is None:
            aux = Node()
            aux.setInfo(n)
            aux.setEsq(None)
            aux.setDir(None)
        elif n < aux.getInfo():
            aux.setEsq(self.inserir(aux.getEsq(), n))
        else:
            aux.setDir(self.inserir(aux.getDir(), n))
        return aux
```

ADT – Árvores Binárias – *Definindo Main*

```
# RECURSIVO

raiz = None
arv = Arvore()
while True:
    n = int(input("Digite o Elemento(tecle 0 para parar): "))
    if n is not 0:
        raiz = arv.inserir(raiz, n)
    else:
        break
```

ADT – Árvores Binárias – *Exercício 01*

Escreva um programa para inserir e em seguida determinar o número de nós numa árvore binária.

ADT – Árvores Binárias – *Exercício 02*

Altere o programa anterior para determinar a soma do conteúdo dos nós numa árvore binária.

ADT – Árvores Binárias – *Exercício 03*

Altere o programa anterior para determinar a média do conteúdo dos nós numa árvore binária.

ADT – Árvores Binárias – *Exercício 04*

Altere o programa anterior para permitir localizar um determinado valor numa árvore binária.

ADT – Árvores Binárias – *Exercício 05*

Altere o programa anterior para permitir a impressão da árvore binária.

```
public void imprime(){    // Em JAVA
     Stack PilhaGeral = new Stack();
     PilhaGeral.push(raiz);
     int nBrancos = 32;
     boolean ehLinhaVazia = false;
     String str = "";
     while(ehLinhaVazia == false){
          Stack PilhaLocal = new Stack();
          ehLinhaVazia = true;
          for(int j=0;j<nBrancos;j++)
               str += " ";
          while(PilhaGeral.isEmpty()==false){
               Node temp = (Node)PilhaGeral.pop();
               if(temp != null){
                    str += temp.info;
                    PilhaLocal.push(temp.esq);
                    PilhaLocal.push(temp.dir);
                    if(temp.esq != null || temp.dir != null)
                         ehLinhaVazia = false;
               }
               else
               {    str += "--";
                    PilhaLocal.push(null);
                    PilhaLocal.push(null);
               }
               for(int j=0;j<nBrancos*2-2;j++)
                    str += ' ';
          }
          str += "\n";
          nBrancos /= 2;
          while(PilhaLocal.isEmpty() == false)
               PilhaGeral.push(PilhaLocal.pop());
     }
     JOptionPane.showMessageDialog(null, str);
}
```

```
def imprime(self, raiz):    # Em PYTHON
        PilhaGeral = []
        PilhaGeral.append(raiz)
        nBrancos = 32
        ehLinhaVazia = False
        saida = ''
        while ehLinhaVazia is False:
            PilhaLocal = []
            ehLinhaVazia = True
            j = 0
            while j < nBrancos:
                saida += " "
                j += 1
            while len(PilhaGeral) is not 0:
                temp = PilhaGeral.pop()
                if temp is not None:
                    saida += str(temp.getInfo())
                    PilhaLocal.append(temp.getEsq())
                    PilhaLocal.append(temp.getDir())
                    if temp.getEsq() is not None or temp.getDir() is not None:
                        ehLinhaVazia = False
                else:
                    saida += "--"
                    PilhaLocal.append(None)
                    PilhaLocal.append(None)
                j = 0
                while j < nBrancos*2-2:
                    saida += ' '
                    j += 1
            saida += "\n"
            nBrancos /= 2
            while len(PilhaLocal) is not 0:
                PilhaGeral.append(PilhaLocal.pop())
        print(saida)
```

ADT – Árvores Binárias – *Exercícios*

06. Altere o programa anterior para permitir a exclusão de um nó na árvore binária.

07. Escreva um programa para determinar a profundidade de uma árvore binária.

08. Escreva um programa recursivo contemplando as funcionalidades das questões 01 a 07.

09. Escreva um programa que solicita "um valor" e retorne VERDADEIRO se esse nó for a raiz de uma árvore binária válida, e FALSO, caso contrário.

10. Escreva um programa que solicita um valor de um nó da árvore, e retorne o nível do nó na árvore.

ADT – Árvores Binárias – *Exercícios*

11. Escreva um programa recursivo para inserir inteiros, excluir e percorrer a árvore binária em ordem, pré-ordem e pós-ordem.

12 ***BÔNUS.*** Faça uma aplicação de árvores destinada à execução de jogos por computador. Escreva um programa para determinar o melhor "movimento" no "jogo-da-velha" a partir de determinada posição do tabuleiro.