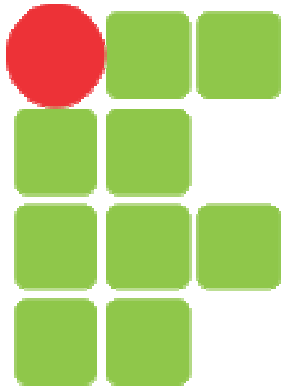


INSTITUTO FEDERAL
CEARÁ

Ernani Andrade Leite

ernani@ifce.edu.br

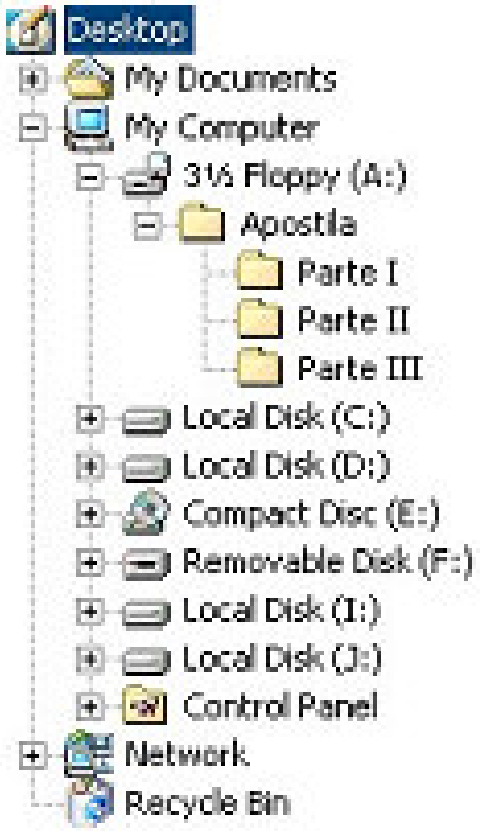
Introdução

- Os arquivos (documentos) que criamos num computador são armazenados dentro de uma estrutura hierárquica de diretórios (pastas).

- Existe um diretório base dentro do qual podemos armazenar diversos sub-diretórios e arquivos. Por sua vez, dentro dos sub-diretórios, podemos armazenar outros sub-diretórios e arquivos, e assim por diante, recursivamente.

Introdução

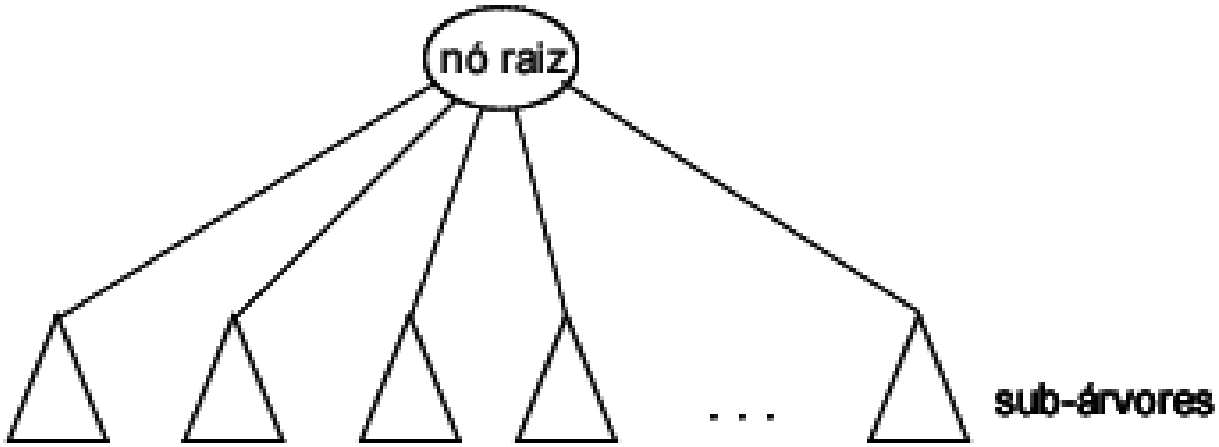
- A Figura abaixo mostra uma imagem de uma árvore de diretórios no Windows.



Conceitos

- Árvores são estruturas de dados adequadas para a representação de hierarquias. A forma mais natural para definirmos uma estrutura de árvore é usando recursividade.
- Uma árvore é composta por um conjunto de nós. Existe um nó **r**, denominado **raiz**, que contém zero ou mais sub-árvores, cujas raízes são ligadas diretamente a **r**. Esses nós raízes das sub-árvores são ditos filhos do nó pai, **r**.
- Nós com filhos são comumente chamados de nós internos e nós que não tem filhos são chamados de ***folhas***, ou nós externos. É tradicional desenhar as árvores com a raiz para cima e folhas para baixo, ao contrário do que seria de se esperar.

Estrutura de Árvore



- Observamos que, por adotarmos essa forma de representação gráfica, não representamos explicitamente a direção dos ponteiros, subentendendo que eles apontam sempre do pai para os filhos.

Estrutura de Árvore

- O número de filhos permitido por nó e as informações armazenadas em cada nó diferenciam os diversos tipos de árvores existentes.
- Estudaremos dois tipos de árvores. Primeiro, examinaremos as árvores binárias, onde cada nó tem, no máximo, dois filhos. Depois examinaremos as chamadas árvores genéricas, onde o número de filhos é indefinido.
- Estruturas recursivas serão usadas como base para o estudo e a implementação das operações com árvores.

Definição (1/2)

Uma árvore é uma estrutura de dados que se caracteriza por uma relação de hierarquia entre os elementos que a compõem. Exemplos de estruturas em forma de árvores são:

- O organograma de uma empresa;
- A divisão de um livro em capítulos, seções, tópicos, etc;
- A árvore genealógica sobre a origem de uma família.

Definição (2/2)

De um modo um pouco mais formal, pode-se dizer que uma árvore é um conjunto finito de um ou mais nodos (nós ou vértices), tais que:

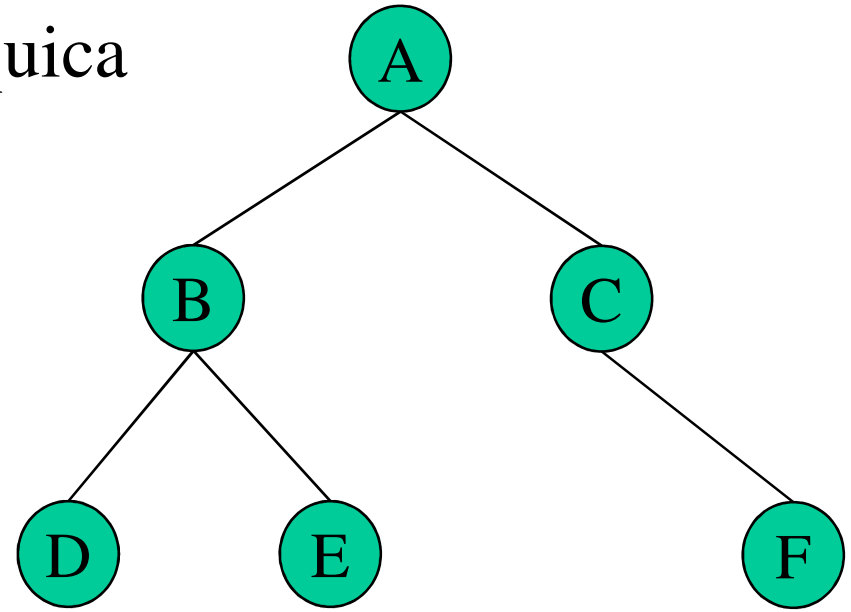
1. existe um nodo denominado **raiz**;
2. os demais nodos formam n >= 0 conjuntos *disjuntos* s_1, s_2, ... , s_n, tais que cada um desses conjuntos também é uma árvore (denominada sub-árvore).

A definição é "recursiva" uma vez que uma árvore é definida em termos de suas sub-árvores, sendo que as sub-árvores também são árvores.

Representações Básicas (1/3)

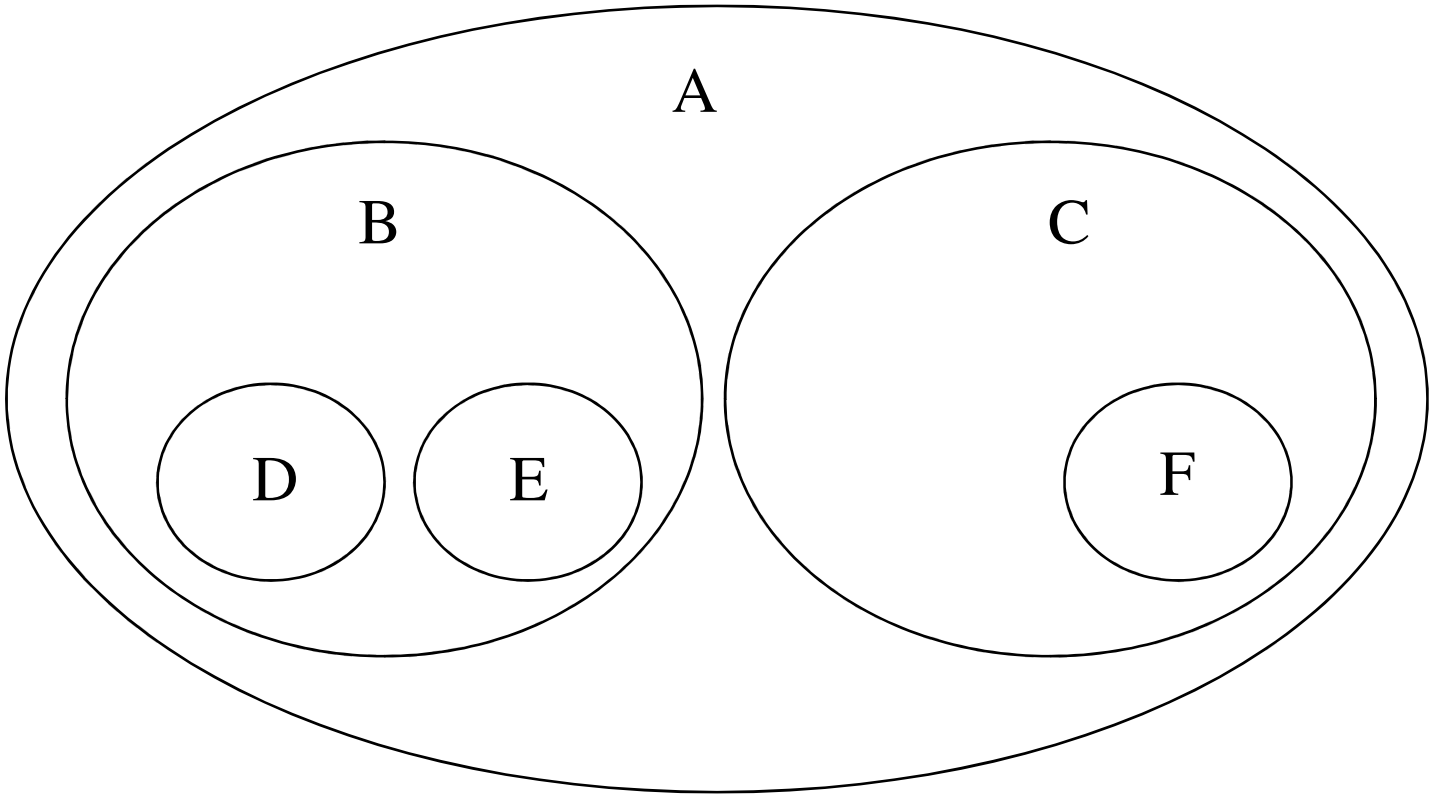
Para visualizar o conceito de árvore, pode-se utilizar de diferentes representações gráficas, como por exemplo:

a) Representação Hierarquica



Representações Básicas (2/3)

b) Representação por conjuntos (diagrama de inclusão)



Representações Básicas (3/3)

c) Representação por expressão parentetizada
(ou parênteses aninhados)

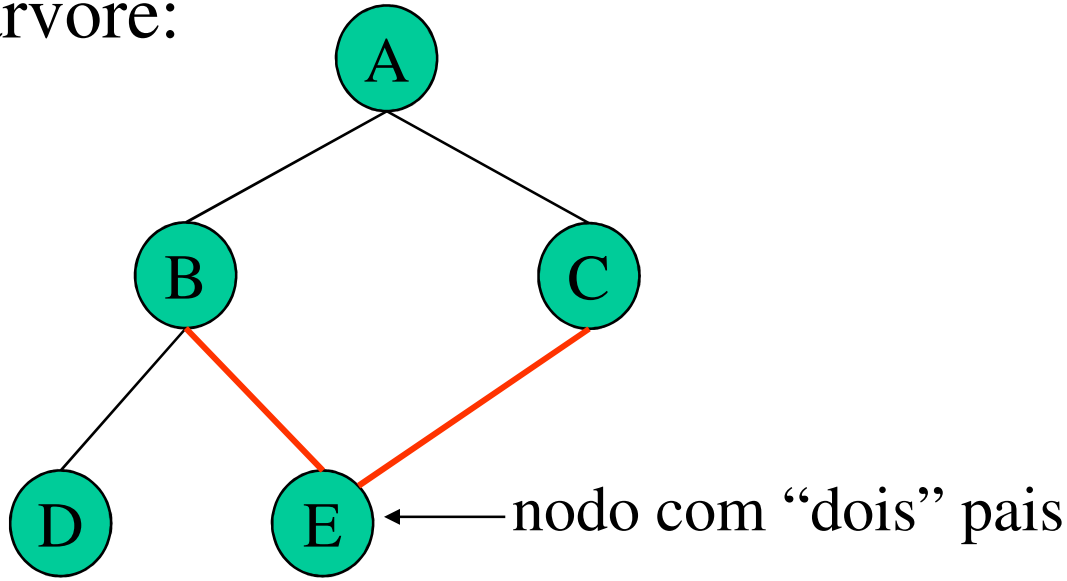
Cada conjunto de parênteses correspondentes contém um nodo e seus filhos.

(A (B (D) (E)) (C (F)))

Aplicando a Definição

Como, por definição, os subconjuntos s_1, s_2,...,s_n são *disjuntos*, cada nó só pode ter um pai.

Assim, o desenho a seguir, por exemplo, "não" representa uma árvore:

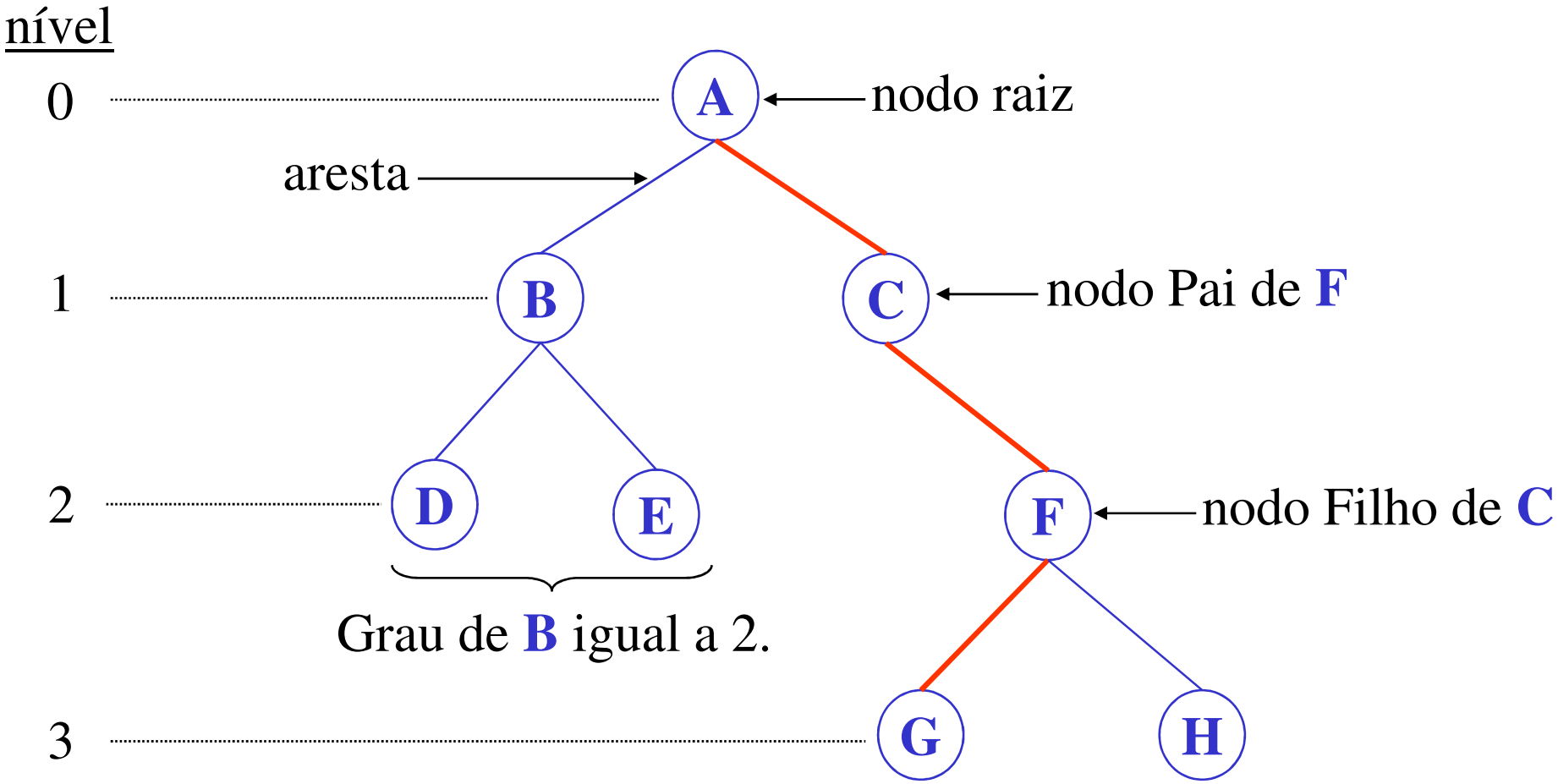


Outras Definições (1/2)

1) A linha que liga dois nodos da árvore denomina-se aresta.
2) Diz-se que existe caminho entre dois nodos V e W da árvore, se a partir do nodo V puder-se chegar ao nodo W percorrendo-se as arestas que ligam os nodos intermediários entre V e W. Observa-se que existe sempre um caminho entre a raiz e qualquer nodo da árvore.
3) Se houver um caminho entre V e W, começando em V diz-se que V é um nodo ancestral de W e W é um nodo descendente de V. Se este caminho contiver uma única aresta, diz-se que V é o nodo pai de W e que W é um nodo filho de V. Dois nodos que são nodos filhos do mesmo nodo pai são denominados nodos irmãos. Uma característica inerente a árvores é que qualquer nodo, exceto a raiz, tem um único nodo pai.

Outras Definições (2/2)

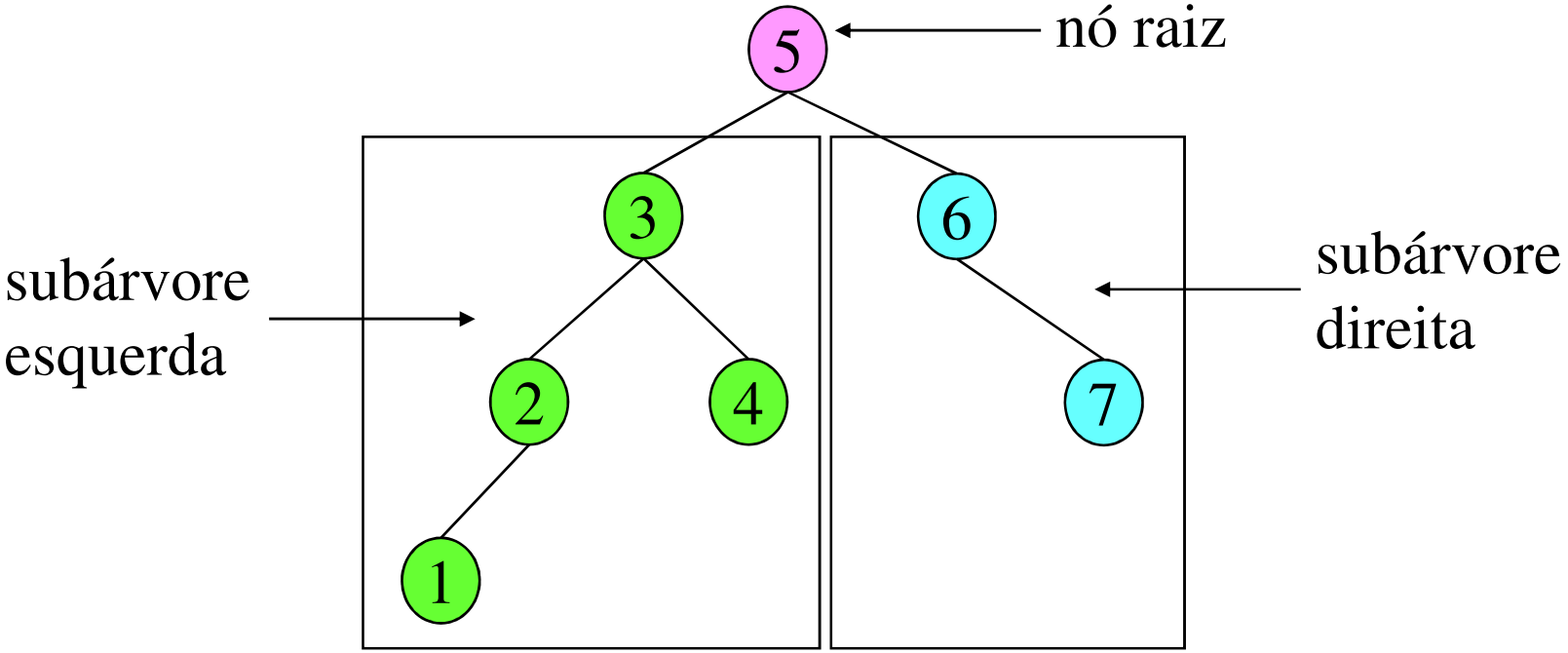
4) Se um nodo não possui nodos descendentes, ele é chamado de folha ou nodo terminal da árvore.
5) Grau de um nodo é o número de nodos filhos do mesmo. Obviamente que um nodo folha tem grau zero.
6) O grau da árvore é igual ao grau do nodo de maior grau da árvore. Grau 2 = Árvore Binária. Grau 3 = Árvore Ternária.
7) Nível de um nodo é o número de nodos existentes no caminho entre a raiz e o próprio nodo. A raiz tem nível 0.
8) O nível da árvore é igual ao nível do nodo de maior nível da árvore.
9) A altura de um nó é o comprimento do caminho mais longo deste nó até o nó folha. A altura de uma árvore é a altura do nó raiz.



Caminho de **A** até **G**: **A**, **C**, **F** e **G** (altura da árvore igual a 4).
D, **E**, **G** e **H** são chamados de folhas ou nós terminais.

Árvore Binária (1/2)

Segundo Knuth (1997, p.312) uma árvore binária é definida como um conjunto finito de nós que ou está vazio ou consiste de um nó chamado de raiz mais os elementos de duas árvores binárias distintas chamadas de subárvore esquerda e subárvore direita do nó raiz.



Árvore Binária (2/2)

Em uma árvore binária, cada nó possui grau zero, um ou dois.

Uma árvore binária é considerada estritramente binária se cada nó da árvore possui grau zero ou dois.

Uma árvore binária é dita “completa” se todo nível i, com exceção do último, tem o número máximo de elementos, ou seja, 2^i.

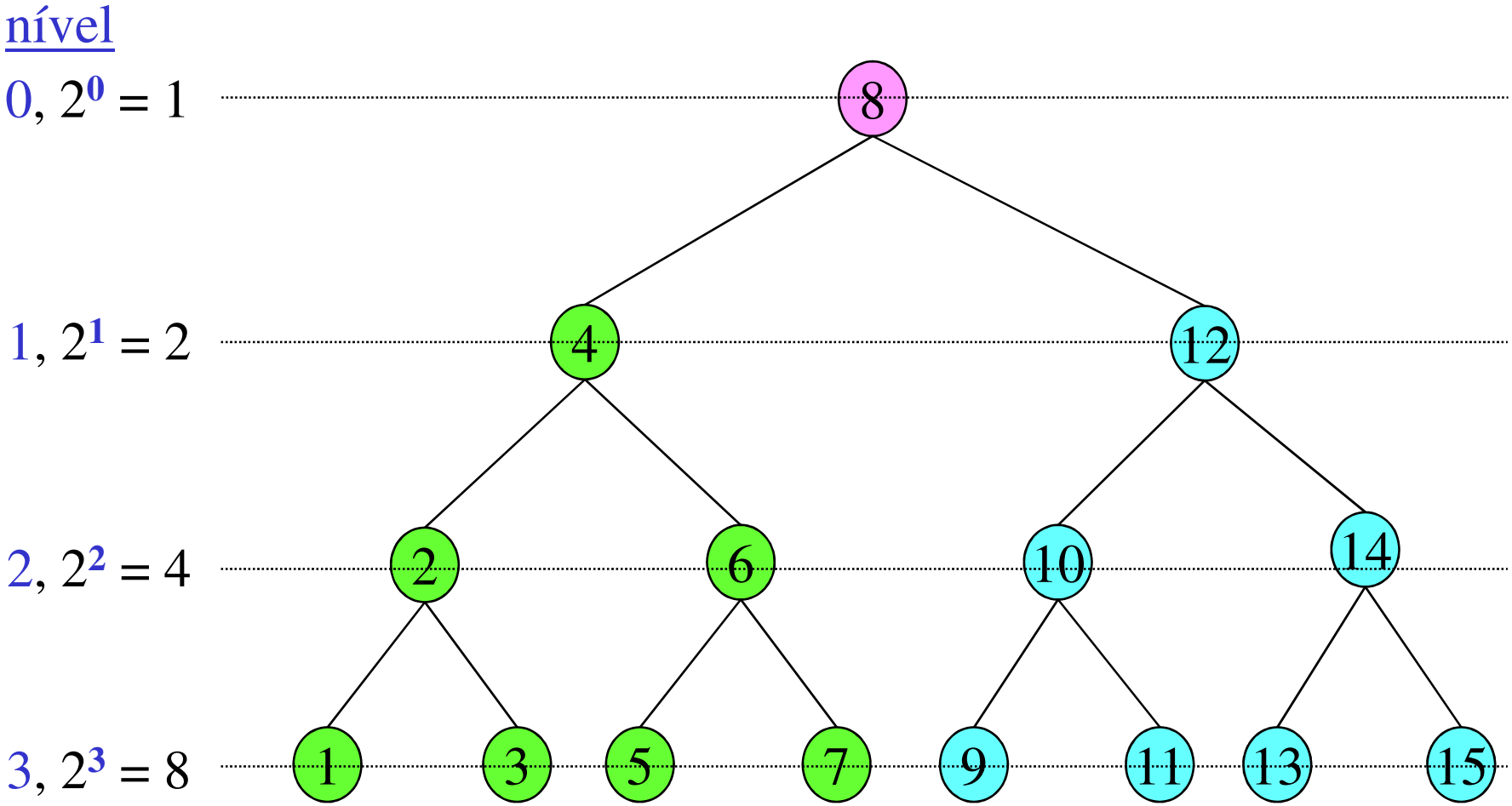
Outras Definições

- Árvore Binária *T* é um conjunto finito de elementos denominados nós ou vértices, tal que:
  - T = 0 e a árvore é dita vazia ou
  - Existe um nó *r*, chamado raiz de *T*, e os nós restantes podem ser divididos em dois subconjuntos disjuntos, *Tre* e *Trd*, que são as sub-árvores esquerda e direita de *r*, respectivamente e as quais, por sua vez, também são árvores binárias.

Vantagens

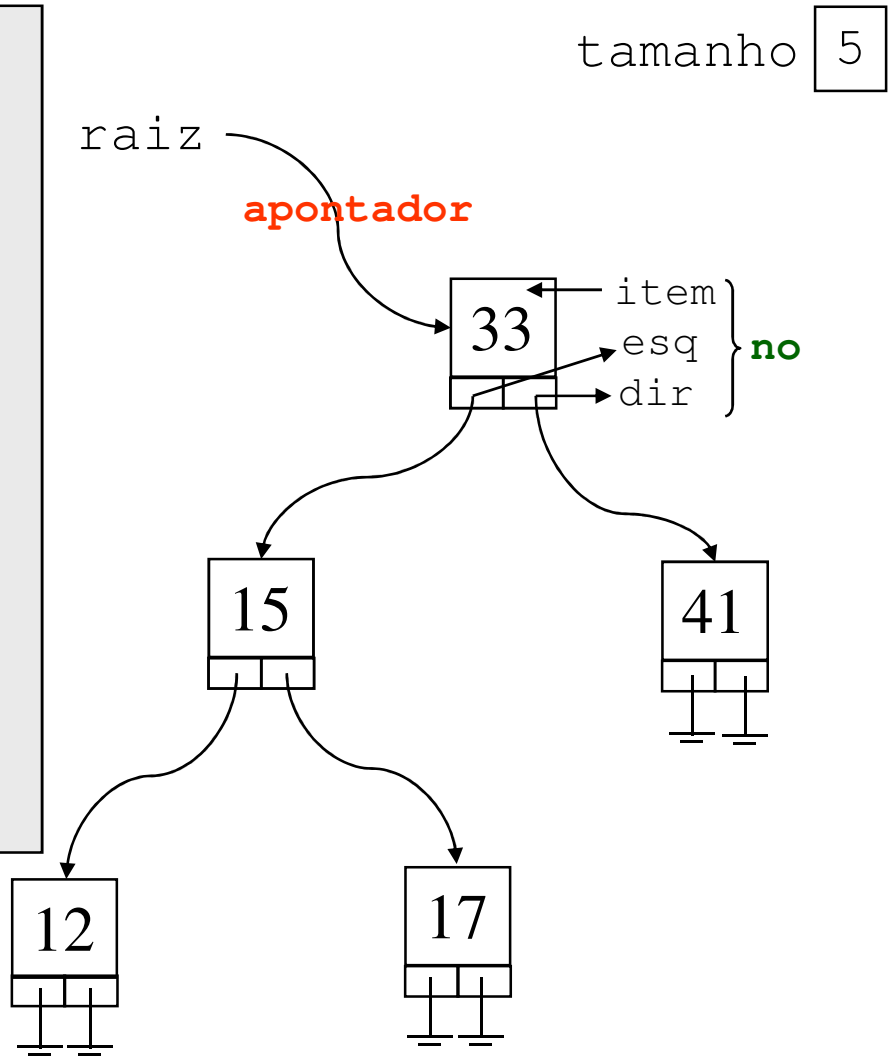
- Possuem um número constante de sub-árvores em cada nó
  - Limitação do número de ponteiros usados
- Algoritmos eficientes para o tratamento
- A forma de armazenar os nós surge naturalmente de sua definição:
  - Ponteiro para o nó raiz (como nas listas lineares)
  - Ponteiros para os filhos: esq e dir
  - Necessita de 2n+1 ponteiros para representar n nós.

Árvore Binária Completa



Implementação de Árvore Binária

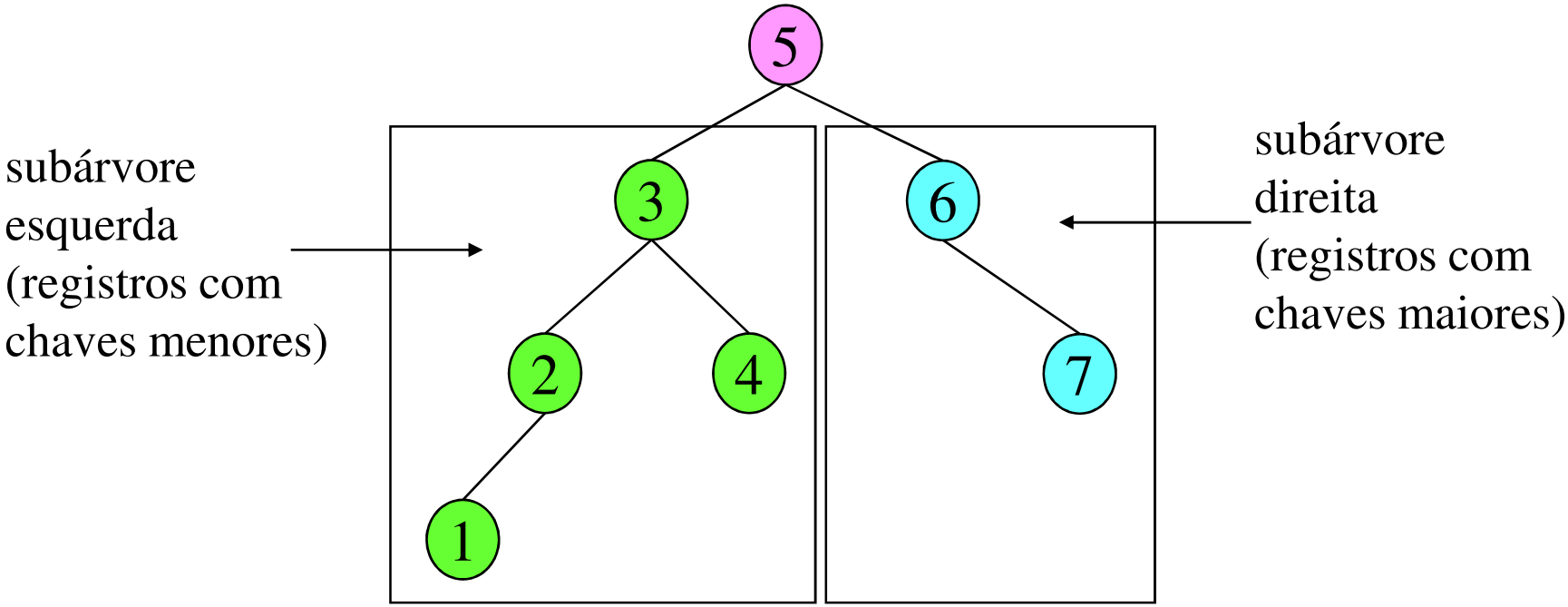
```
struct tipoItem {
  int chave;
};

typedef struct no *apontador;

struct no {
  struct tipoItem item;
  apontador esq;
  apontador dir;
};

struct tipoArvore {
  apontador raiz;
  int tamanho;
};
```



Árvore Binária de Pesquisa

Uma árvore binária de pesquisa é uma árvore binária em que todo nó não-terminal (ou interno) contém um registro, e, para cada nó, a seguinte propriedade é verdadeira:

todos os registros com chaves menores estão na subárvore esquerda e todos os registros com chaves maiores estão na subárvore direita.



Procedimento para Inserir um nó na Árvore Binária de Pesquisa

```
void insere(tipoItem x, apontador *p) {
  if ((*p) == NULL) {
     *p = (apontador) malloc(sizeof(no));
     (*p)->item = x;
     (*p)->esq = NULL;
     (*p)->dir = NULL;
  }
  else
     if (x.chave < (*p)->item.chave)
        insere(x, &((*p)->esq));
     else
        insere(x, &((*p)->dir));
}
```

Procedimento para Localizar um Item na Árvore Binária de Pesquisa

```
void pesquisa(tipoItem x, apontador p,
  int *encontrou) {

if ((*p) == NULL)
   *encontrou = 0
else
   if (x.chave < (p*)->item.chave)
// pesquisa na subárvore esquerda
      pesquisa(x, (p*)->esq, encontrou);
   else
      if (x.chave > (p*)->item.chave)
// pesquisa na subárvore direita
         pesquisa(x, (*p)->dir, encontrou);
      else
         *encontrou = 1;
}
```

Procedimento para Retirar um nó na Árvore Binária de Pesquisa

```
void antecessor(apontador q,
  apontador *r) {
  if ((*r)->dir != NULL)
     antecessor(q, &((*r)->dir));
  else {
     q->item = (*r)->item;
     q = *r;
     *r = (*r)->esq;
     free(q);
  }
}
```

se o nó que contém o item a ser retirado possui no máximo um descendente

se o nó que contém o item a ser retirado conter dois descendentes, o item a ser retirado dever ser primeiro substituido pelo item mais à direita na subárvore a esquerda, ou pelo item mais à esquerda na subárvore da direita.

```
void retira(tipoItem x, apontador *p, int *operacao) {
  apontador aux;
  if ((*p) == NULL)
     *operacao = 0;
  else
     if (x.chave < (*p)->item.chave)
        retira(x, (&(*p)->esq), operacao);
     else
        if (x.chave > (*p)->item.chave)
           retira(x, (&(*p)->dir), operacao);
        else {
           *operacao = 1;
           if ((*p)->dir == NULL) {
              aux = *p;
              *p = (*p)->esq;
              free(aux);
           }
           else
              if ((*p)->esq == NULL) {
                 aux = *p;
                 *p = (*p)->dir;
                 free(aux);
              }
              else
                 antecessor((*p), (&(*p)->esq));
        }
}
```

Caminhamento em Árvore Binária de Pesquisa

Em geral são utilizadas três formas de caminhamentos em árvores binárias de pesquisa e estes são determinados dependendo da ordem em que são visitados o nodo raiz, sua subárvore à esquerda e sua subárvore à direita.

O termo “visitar” significa a realização de alguma operação sobre a informação do nó, como modificação da mesma, impressão ou qualquer outra.

Existem três tipos de caminhamentos (ou percursos): Percurso em **pré-ordem**, **em-ordem** e **pós-ordem**.

Caminhamento Pré-ordem (1/2)

Ou ainda, caminhamento pré-fixado (**R E D**):

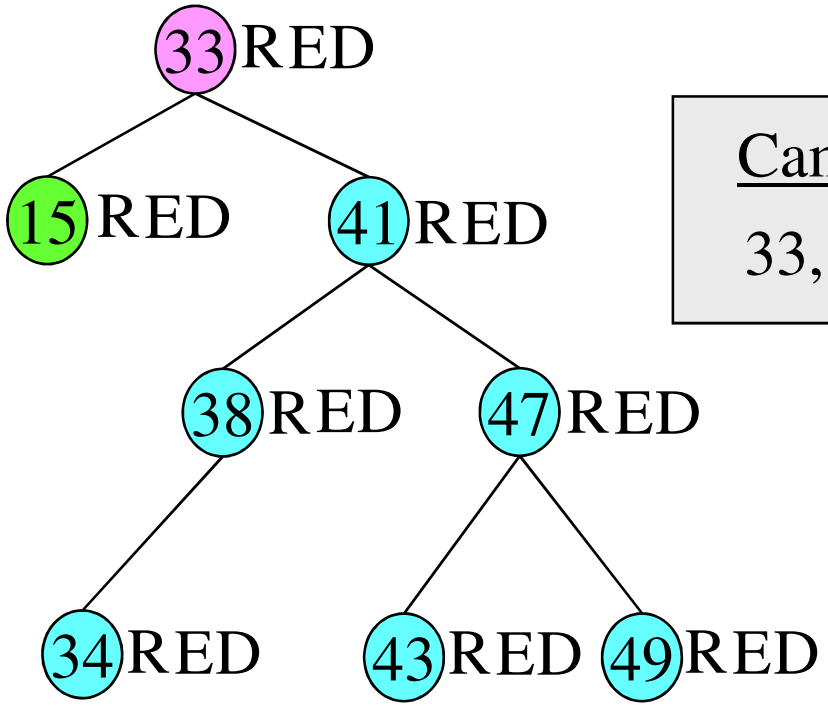
1. visita a raiz
2. caminha na subárvore esquerda na pré-ordem
3. caminha na subárvore direita na pré-ordem

```
void pre_ordem(apontador p) {
   if (p != NULL) {
      visitaRaiz(p);
      pre_ordem(p->esq);
      pre_ordem(p->dir);
   }
}
```

Caminhamento Pré-ordem (2/2)

Ou ainda, caminhamento pré-fixado (**R E D**):

1. visita a raiz
2. caminha na subárvore esquerda na pré-ordem
3. caminha na subárvore direita na pré-ordem



Caminhamento Pré-Ordem (**R E D**):

33, 15, 41, 38, 34, 47, 43, 49

Caminhamento Em-ordem (1/2)

Ou ainda, caminhamento central (infixado) (**E R D**):

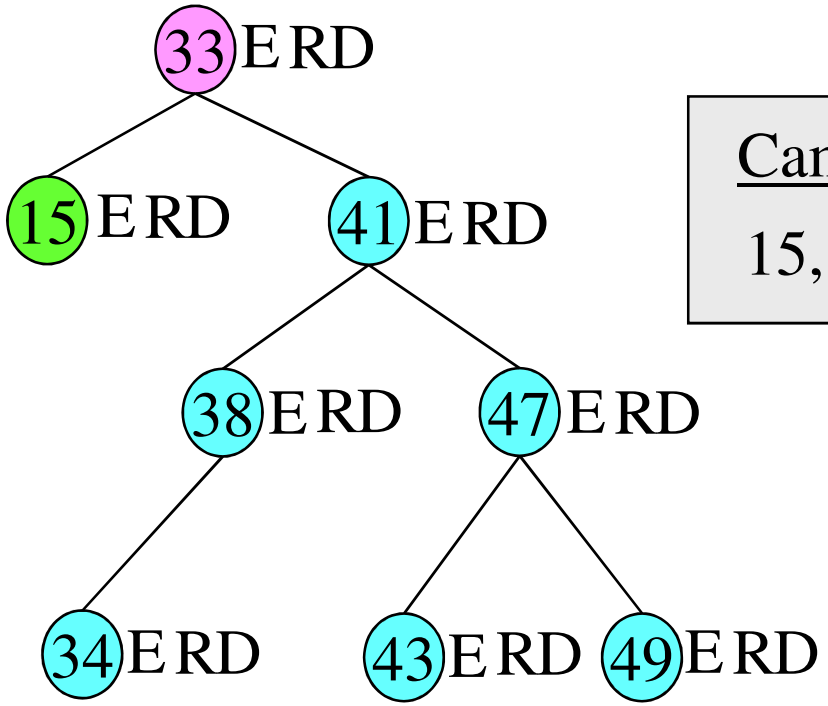
1. caminha na subárvore esquerda na ordem central
2. visita a raiz
3. caminha na subárvore direita na ordem central

```
void em_ordem(apontador p) {
   if (p != NULL) {
      em_ordem(p->esq);
      visitaRaiz(p);
      em_ordem(p->dir);
   }
}
```

Caminhamento Em-ordem (2/2)

Ou ainda, caminhamento central (infixado) (**E R D**):

1. caminha na subárvore esquerda na ordem central
2. visita a raiz
3. caminha na subárvore direita na ordem central



Caminhamento Em-Ordem (**E R D**):

15, 33, 34, 38, 41, 43, 47, 49

Caminhamento Pós-ordem (1/2)

Ou ainda, caminhamento pós-fixado (**E D R**):

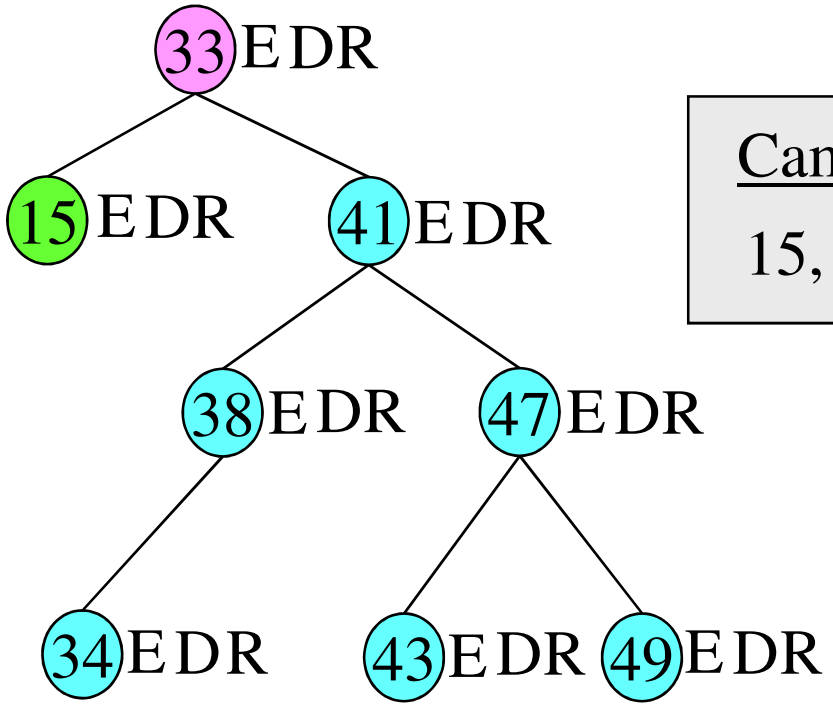
1. caminha na subárvore esquerda na pós-ordem
2. caminha na subárvore direita na pós-ordem
3. visita a raiz

```
void pos_ordem(apontador p) {
   if (p != NULL) {
      pos_ordem(p->esq);
      pos_ordem(p->dir);
      visitaRaiz(p);
   }
}
```

Caminhamento Pós-ordem (2/2)

Ou ainda, caminhamento pós-fixado (**E D R**):

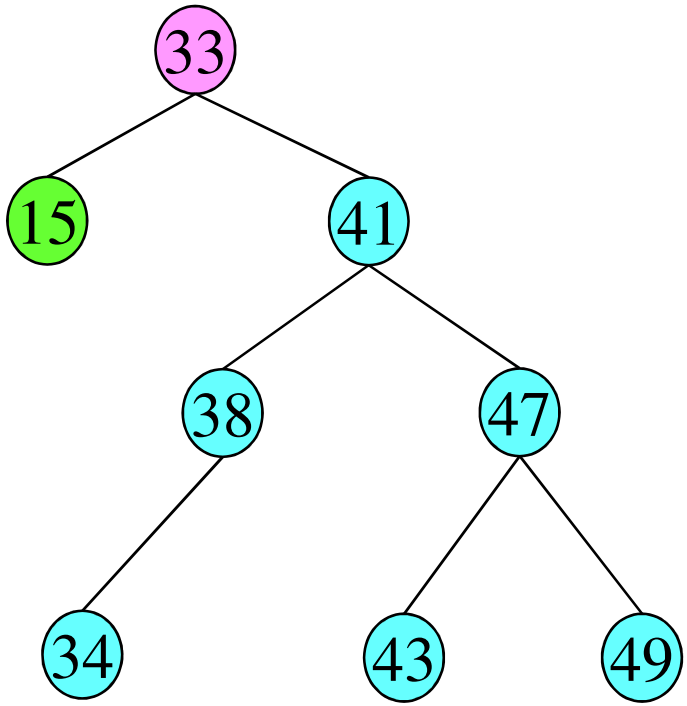
1. caminha na subárvore esquerda na pós-ordem
2. caminha na subárvore direita na pós-ordem
3. visita a raiz



Caminhamento Pós-Ordem (**E D R**):

15, 34, 38, 43, 49, 47, 41, 33

Exercitando os Caminhamentos



Caminhamento Pré-Ordem (**R E D**):
33, 15, 41, 38, 34, 47, 43, 49

Caminhamento Em-Ordem (**E R D**):
15, 33, 34, 38, 41, 43, 47, 49

obs. no caminhamento central os nós são visitados obedecendo a ordem ascendente das chaves.

Caminhamento Pós-Ordem (**E D R**):
15, 34, 38, 43, 49, 47, 41, 33

Veja o Programa Fonte: Arvores.html

Referência

- Projeto de Algoritmos com implementações em Pascal e C
  - Nivio Ziviani
  - São Paulo: Pioneira Thomson Learning, 2005.
  - Capítulo 5: Pesquisa em Memória Primária.
- Eduardo Stefani