Estruturas de Dados I

Profa. Juliana Pinheiro Campos
jupcampos@gmail.com

ESTRUTURAS DE DADOS

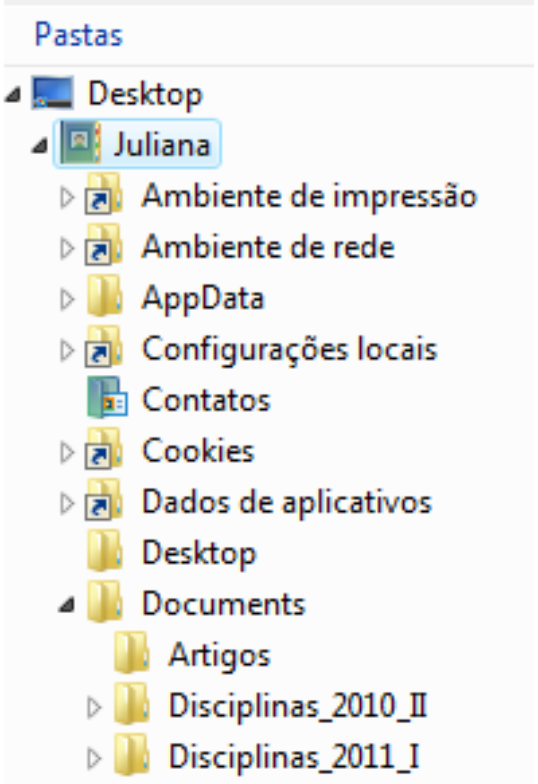
- Árvores
  - Conceitos
  - Árvores binárias
  - Árvores binárias de pesquisa
  - Árvores binárias balanceadas

ESTRUTURAS DE DADOS

Árvores

- Estruturas de dados adequadas para a representação de **hierarquias**.

- Exemplo: sistemas de arquivos em um computador (armazenados em diretórios e subdiretórios);



ESTRUTURAS DE DADOS

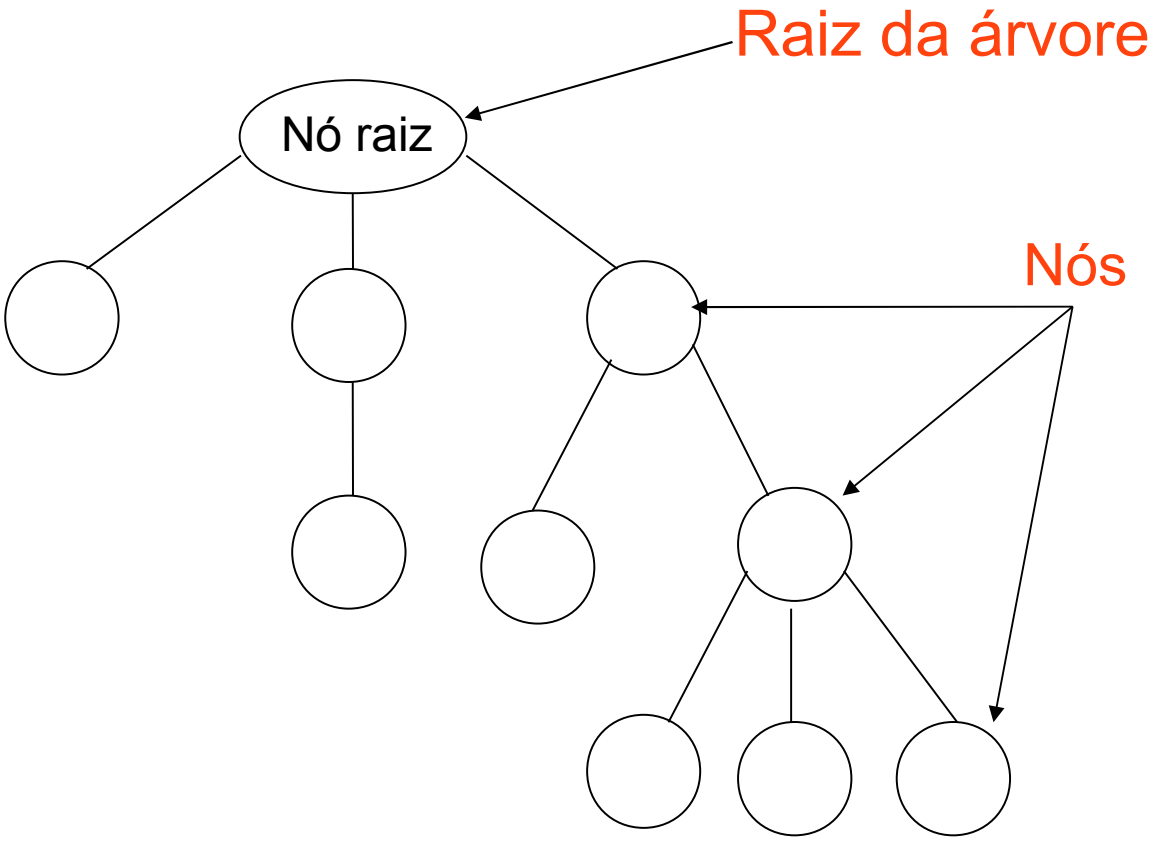
Árvores

- Uma **árvore A** é um conjunto finito de 0 ou mais elementos denominados **nós**.

- Existe um nó **r** denominado **raiz** da árvore.

ESTRUTURAS DE DADOS

- Árvores



Árvores

- A forma mais natural de definir uma estrutura de árvore é usando a **recursividade**.

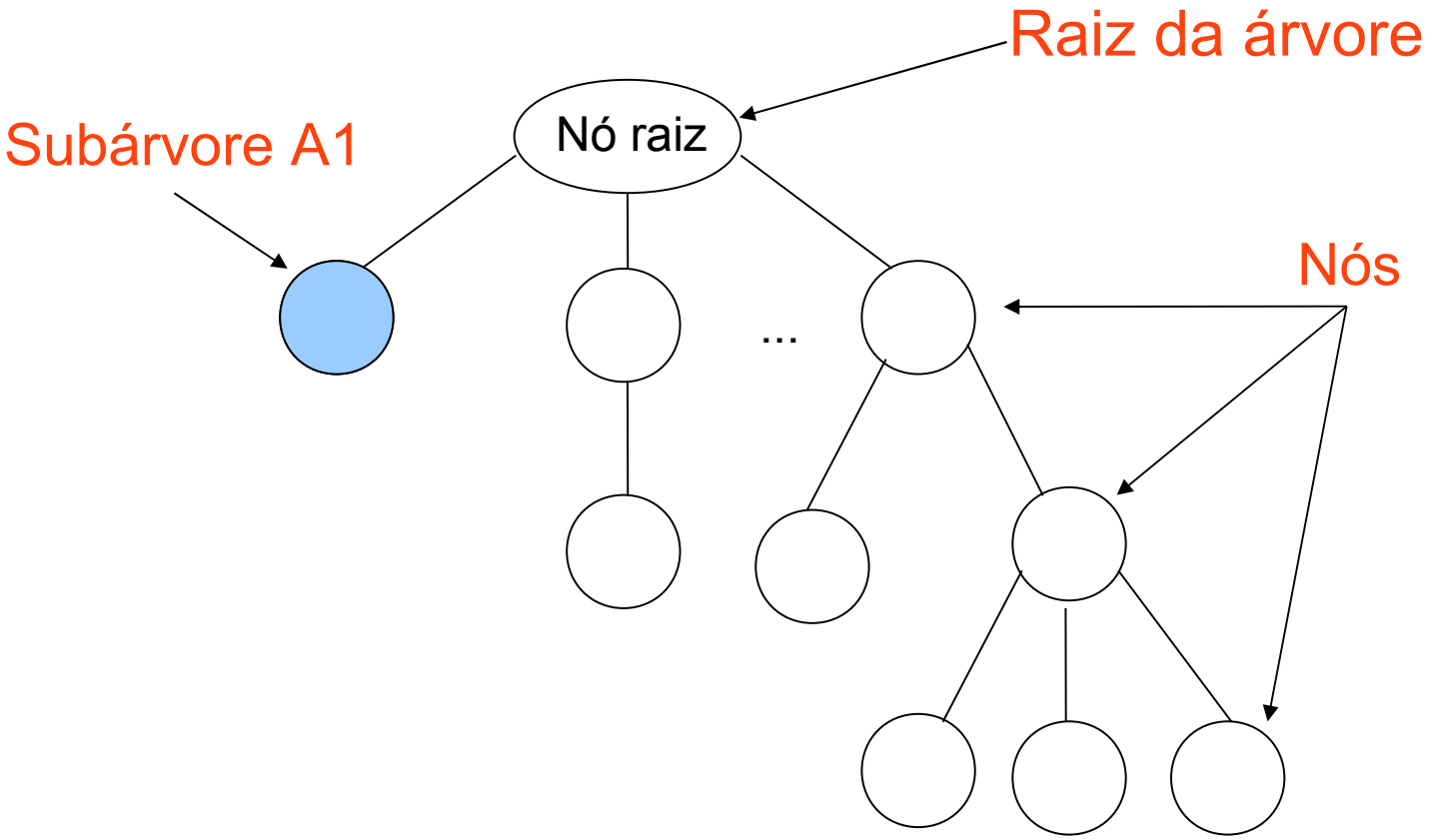
- O nó r (raiz da árvore) contém 0 ou mais **subárvores**, cujas raízes são ligadas diretamente a r.

- Cada nó ligado diretamente a r é raiz de uma subárvore.

Todo nó de uma árvore, é raiz de uma subárvore.

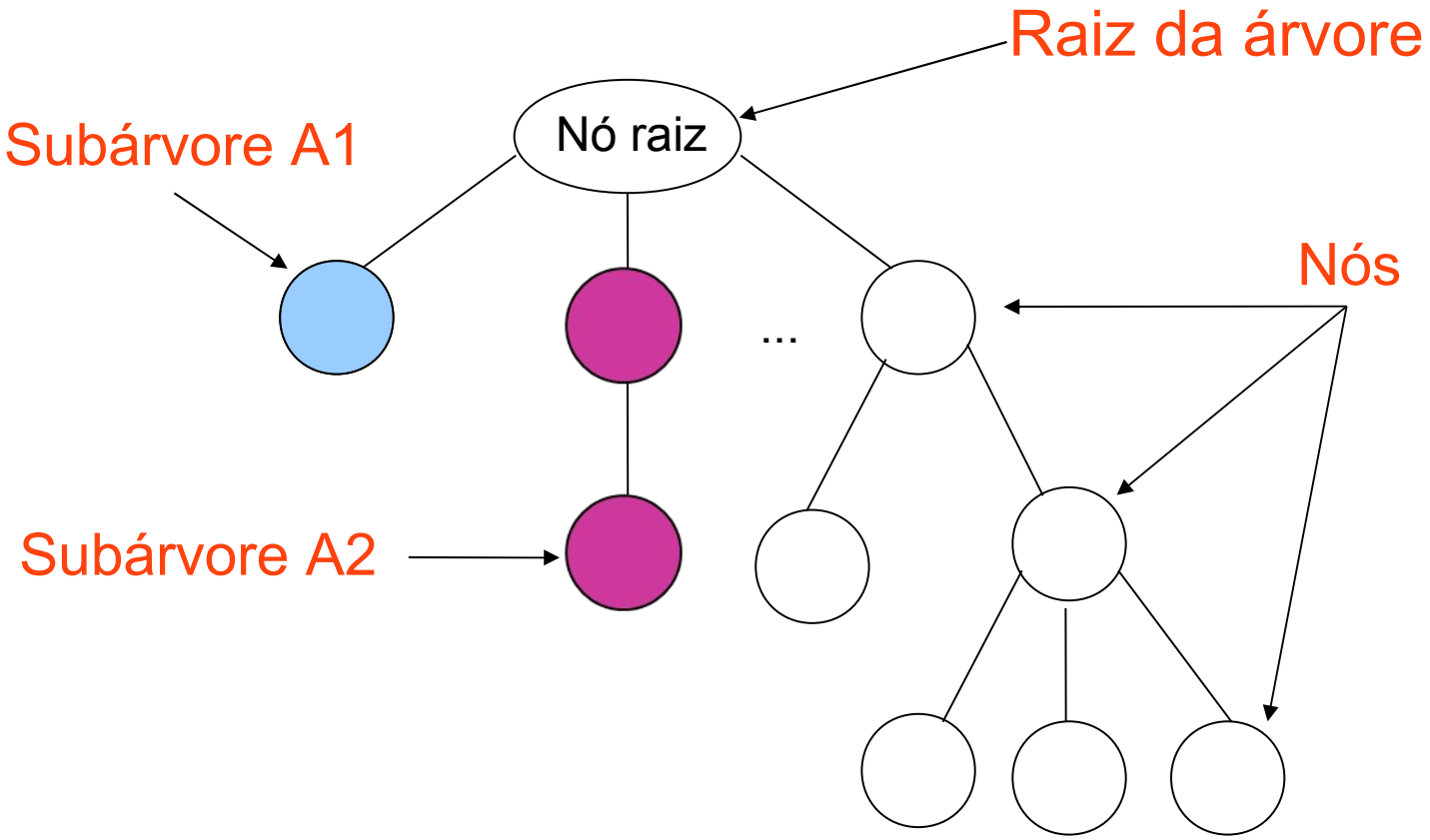
ESTRUTURAS DE DADOS

- Árvores



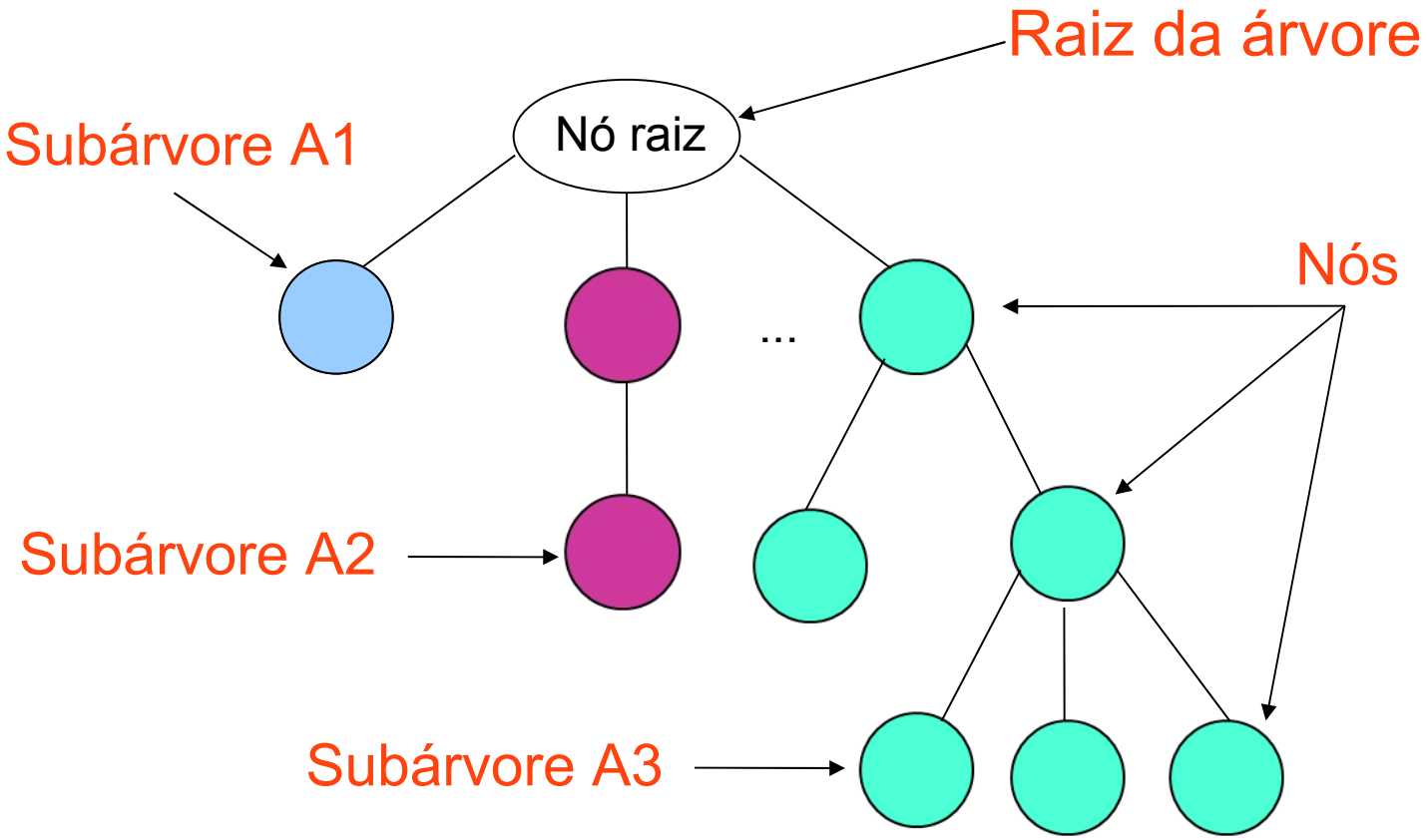
ESTRUTURAS DE DADOS

- Árvores



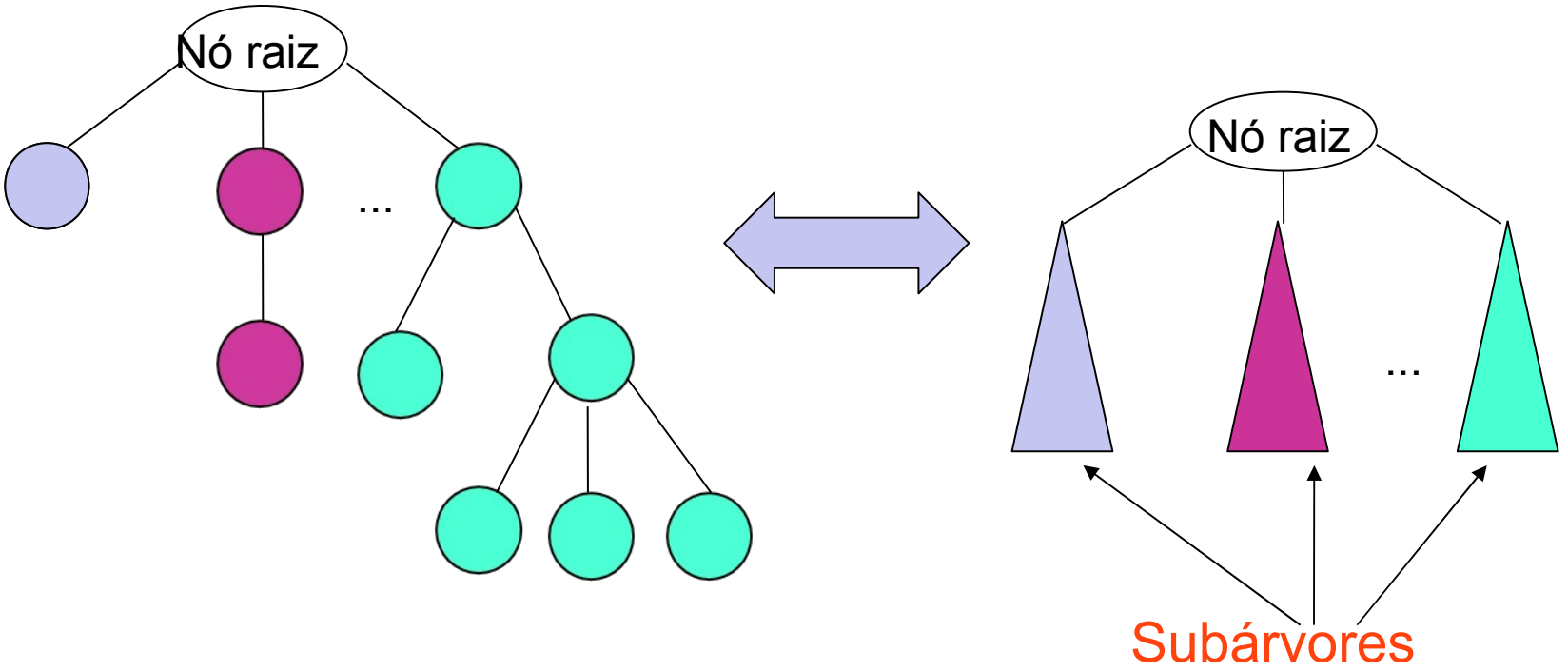
ESTRUTURAS DE DADOS

- Árvores



- Árvores

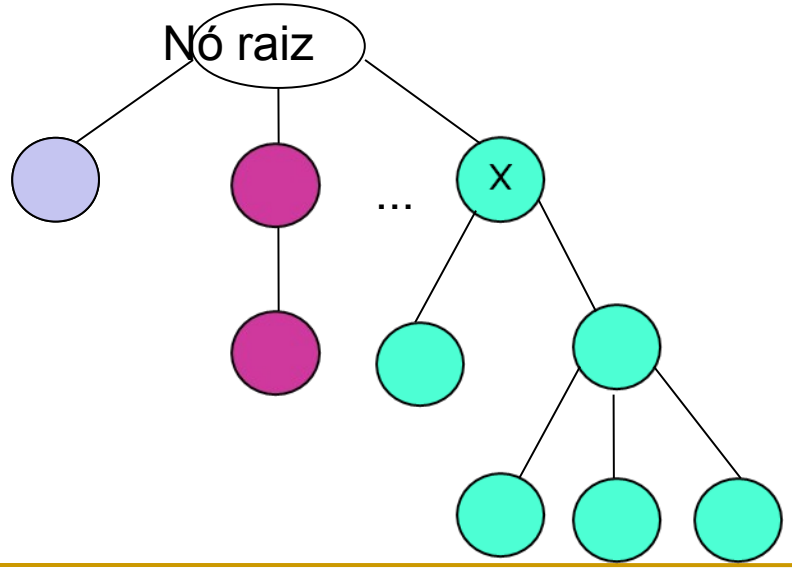
Estrutura de árvore



Árvores

- O número de subárvores de um nó é denominado **grau** do nó.

- O grau da árvore é definido como o maior grau dentre todos os seus nós.



Grau do nó raiz = 3

Grau do nó x = 2

Grau da árvore = 3

Árvores

- Uma árvore com 0 (zero) nós é uma **árvore vazia**.

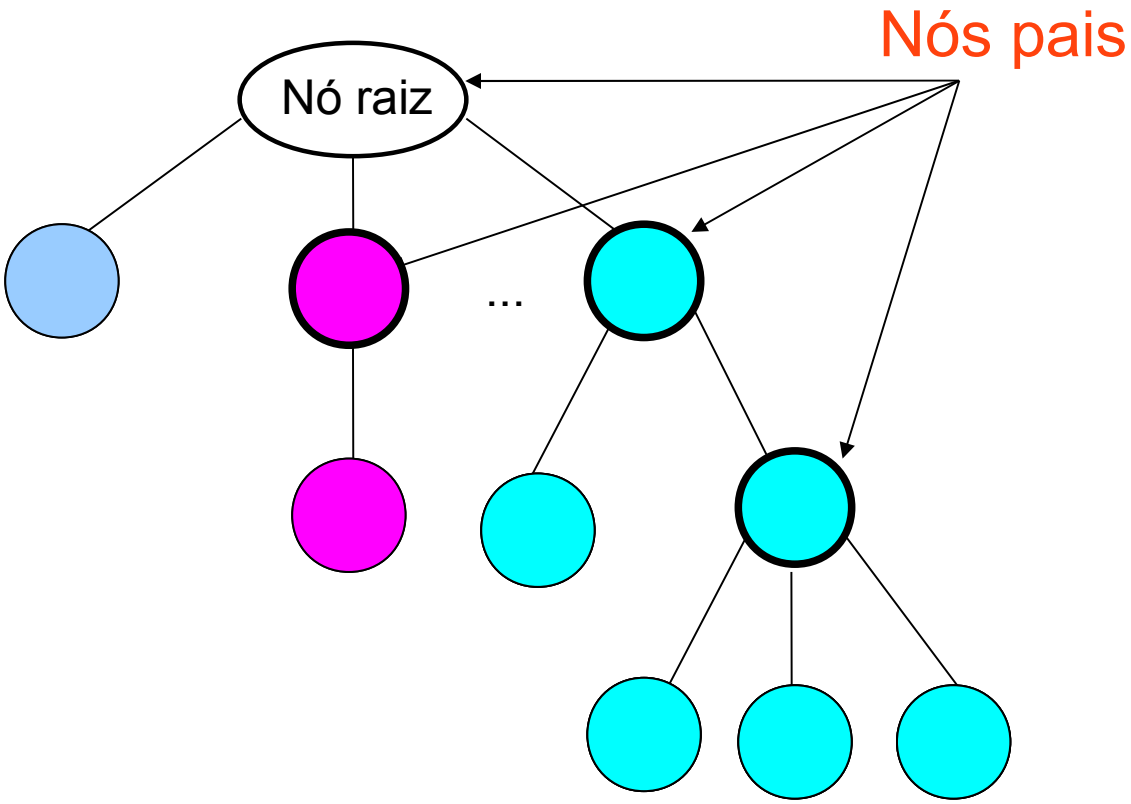
- Os nós raízes das subárvores são ditos **filhos** do nó pai (r);

- **Nós internos**: nós que possuem filhos (nós de grau > 0)

- **Nós externos (folhas)**: nós que não possuem filhos (nós de grau 0).

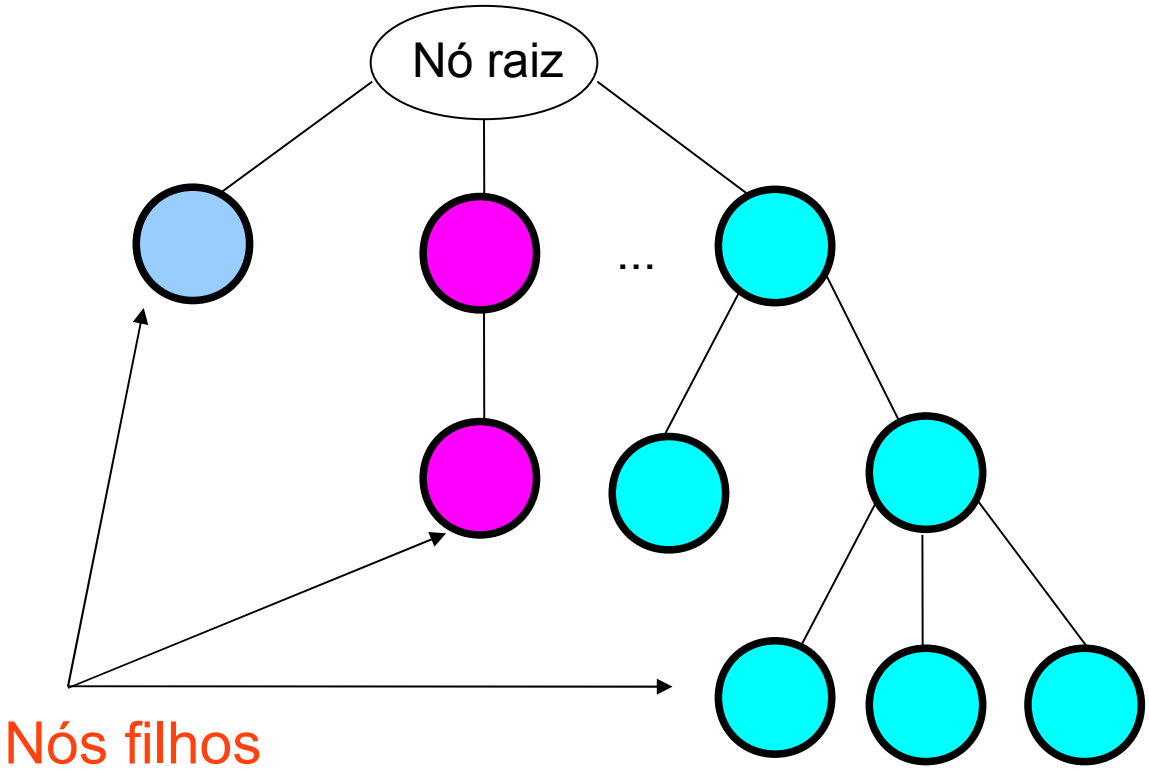
ESTRUTURAS DE DADOS

- Árvores



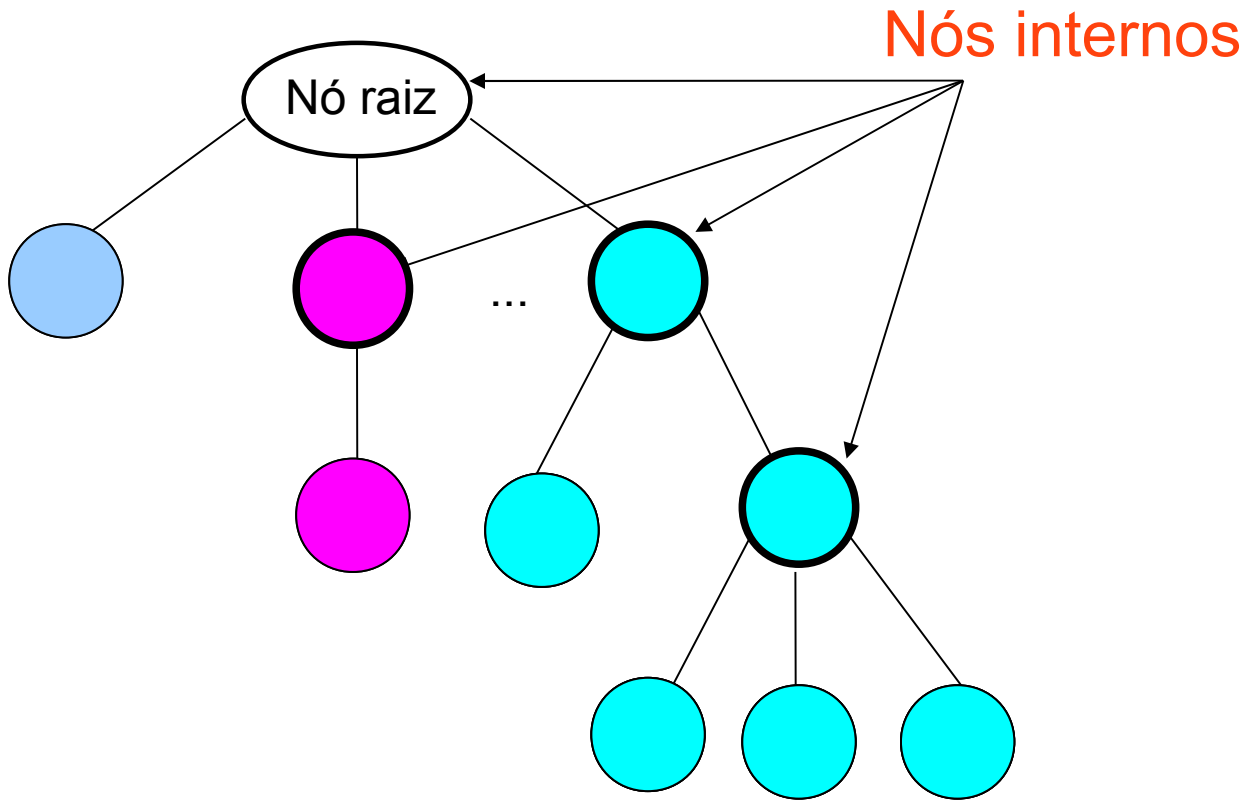
ESTRUTURAS DE DADOS

- Árvores



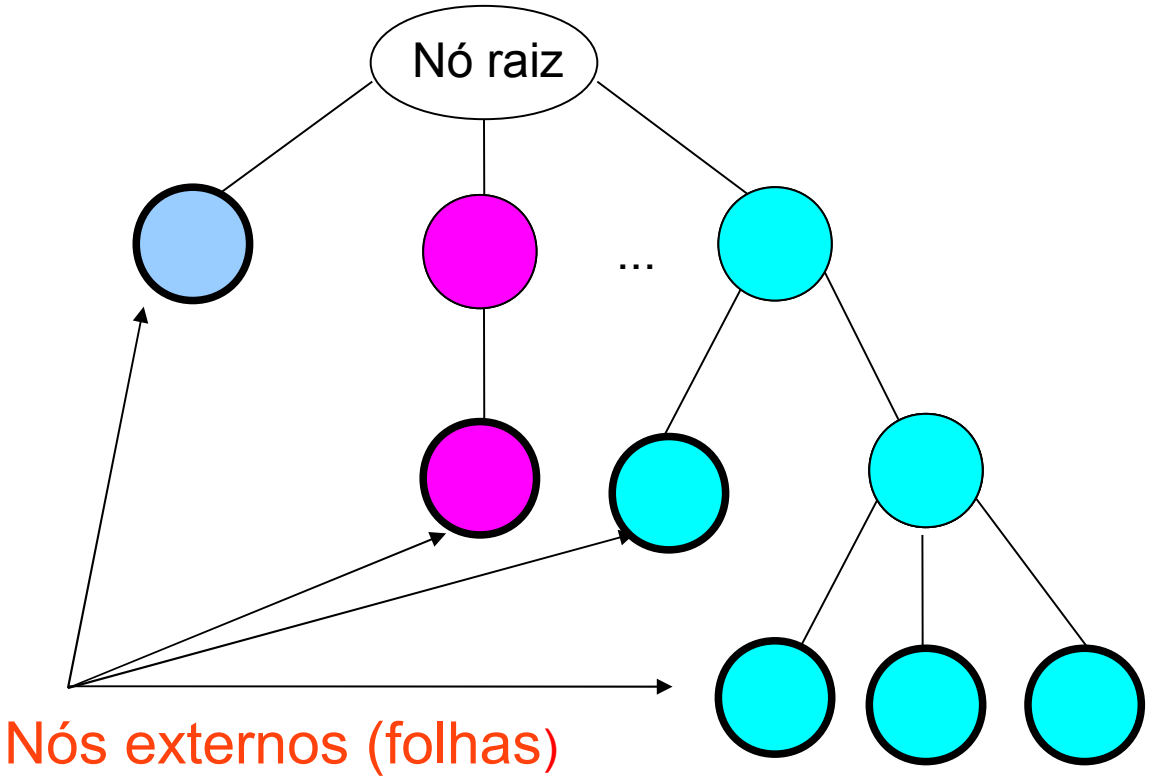
ESTRUTURAS DE DADOS

- Árvores



ESTRUTURAS DE DADOS

- Árvores

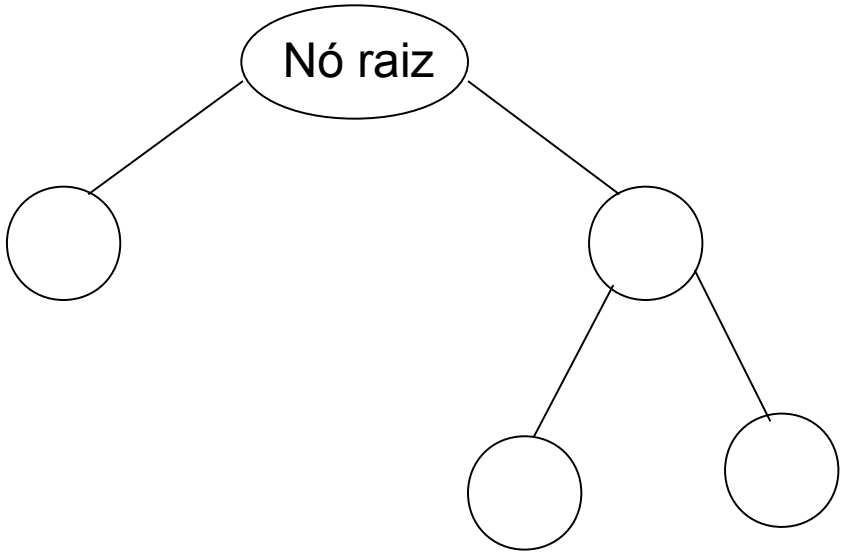


ESTRUTURAS DE DADOS

- Árvores
  - Em uma árvore **nunca pode ocorrer ciclo**.
  - O número de filhos permitido por nó e as informações armazenadas em cada nó diferenciam os vários tipos de árvores existentes.

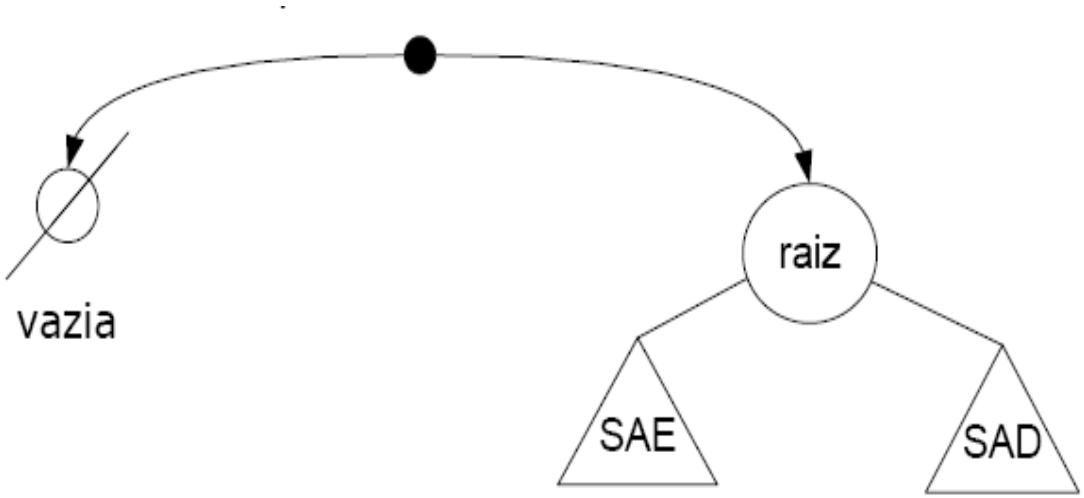
Árvores binárias

- Em uma árvore binária, cada nó tem 0 (zero), 1 (um) ou 2 (dois) filhos.
  - Todos os nós de uma árvore binária tem grau menor ou igual a 2.



Árvores binárias

- Definição recursiva de uma árvore binária:
  - Uma árvore vazia; ou
  - Um nó raiz tendo duas subárvores, identificadas como subárvore da direita (SAD) e a subárvore da esquerda (SAE).

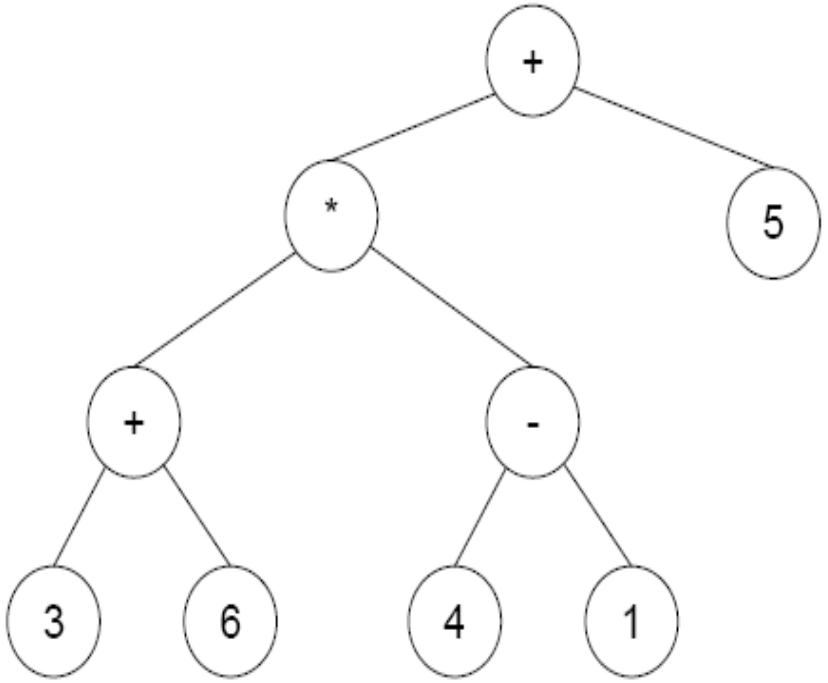


Árvores binárias

- Exemplo de utilização de árvores binárias: avaliação de expressões aritméticas.
  - Os nós folhas representam os operandos.
  - Os nós internos representam os operadores.

Expressão:

(3 + 6) * (4 - 1) + 5

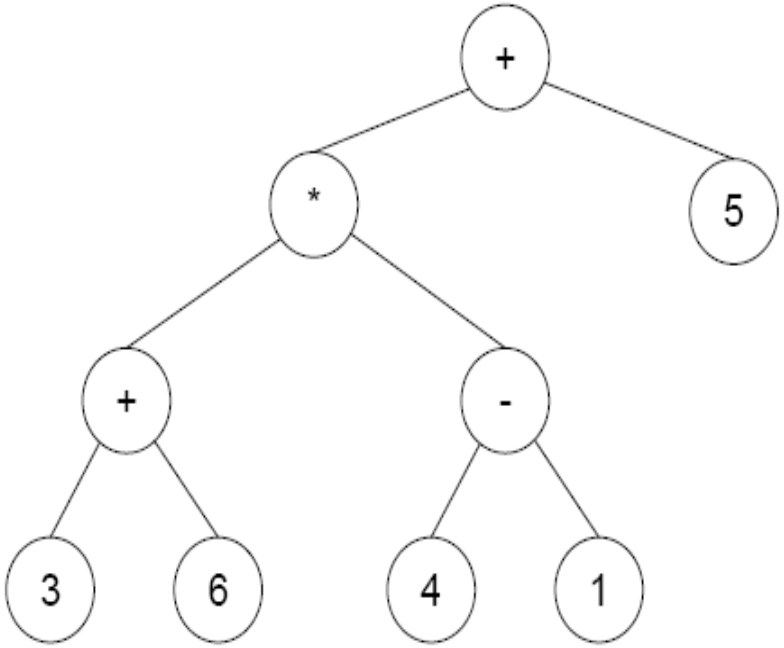


Árvores binárias

- Exemplo de utilização de árvores binárias: avaliação de expressões.

Expressão:

(3 + 6) * (4 - 1) + 5



Raiz: Nó +

Expressão = (SAE Nó +) + (SAD Nó +)

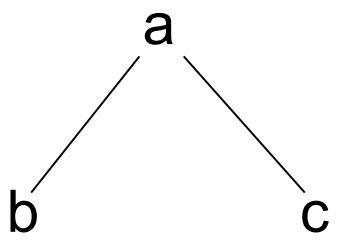
= ((SAE Nó *) * (SAD Nó *)) + 5

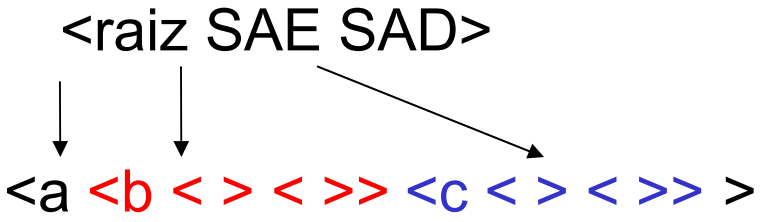
= (((SAE Nó +) + (SAD Nó +)) * ((SAE Nó -) –
(SAD Nó -))) + 5

= (3 + 6) * (4 - 1) + 5

Árvores binárias

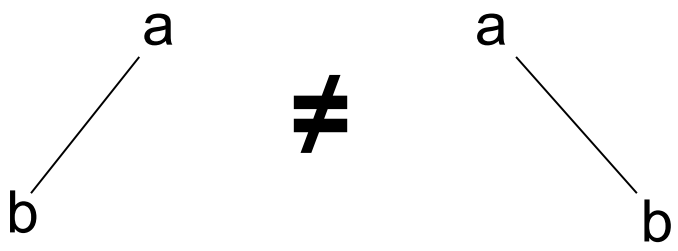
- Notação textual para descrever árvores binárias:
  - Árvore vazia: < >
  - Árvores não vazias: <raiz SAE SAD>





Árvores binárias

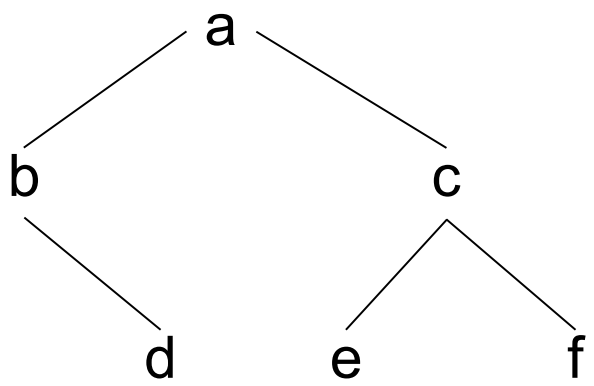
- Uma subárvore é especificada como sendo a SAE ou SAD de uma árvore maior, e qualquer uma das duas subárvores pode ser vazia.



As árvores acima são distintas!

Árvores binárias

- Qual a notação textual descreve a árvore binária abaixo?



<a <b <> <d <> <>>> <c <e <> <>> <f <> <> > > >

Árvores binárias

- Implementação de uma árvore binária em C para armazenar caracteres.
- As funções que manipulam árvores são, em geral, **implementadas de forma recursiva**, por meio da definição recursiva da estrutura.
- Que estrutura podemos usar para representar um nó da árvore?

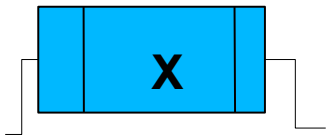
ESTRUTURAS DE DADOS

Árvores binárias

- Estrutura para representar um nó da árvore binária:

```
struct arvore {
  char info;                    // armazena a informação
  struct arvore* esq;           // ponteiro para sae
  struct arvore* dir;           // ponteiro para sad
};

typedef struct arvore Arvore;
```



- A estrutura da árvore é representada por um ponteiro para o nó raiz.

Árvores binárias

Operações que serão implementadas:

Função para criar uma árvore vazia.

Função para criar uma árvore não vazia dadas a informação e a sae e a sad.

Função que verifica se uma árvore está vazia.

Função que verifica se uma determinada informação está armazenada (pertence) à árvore.

Função para imprimir as informações contidas na árvore.

Função para liberar memória alocada dinamicamente.

Árvores binárias

Interface do tipo árvore:

```
Arvore* criarArvoreVazia();

Arvore* criarArvore(char c, Arvore* sae, Arvore* sad);

int estaVazia(Arvore* arv);

int pertenceArvore(Arvore* arv, char c);

void imprimeArvore(Arvore* arv);

Arvore* liberarArvore(Arvore* arv);
```

- Árvores binárias
  - Função que cria uma árvore vazia:

```
Arvore* criarArvoreVazia() {
  return NULL;
}
```

Árvores binárias

- Função que cria uma árvore não vazia dadas a informação que será armazenada e as suas subárvores (sae e sad):

```
Arvore* criarArvore(char c, Arvore* sae, Arvore* sad){
  Arvore* a = malloc(sizeof(Arvore));
  a->info = c;
  a->esq = sae;
  a->dir = sad;
  return a;
}
```

- Árvores binárias
  - Essas duas funções de criação representam os dois casos da definição recursiva de árvore binária.
    - Uma árvore binária (Arvore* a;) é vazia quando a = criarArvoreVazia();
    - Uma árvore binária (Arvore* a;) é não vazia quando a = criarArvore(c, sae, sad);
  - A partir dessas duas funções é possível criar árvores mais complexas.

- Árvores binárias

  - Usando as operações criarArvoreVazia e criarArvore, crie uma estrutura que represente a árvore descrita pela notação textual abaixo:

<a <b <> <d <> <>>> <c <e <> <>> <f <> <> > > >

- Árvores binárias
  - Função que verifica se a árvore está vazia

```
int estaVazia(Arvore* arv){

  return (arv == NULL);

}
```

Árvores binárias

- A função que imprime a árvore percorre todos os nós imprimindo sua informação.
- Como ela pode ser implementada recursivamente?
- Se a árvore não é vazia, imprimimos a informação associada à raiz e chamamos (recursivamente) a função para imprimir as subárvores.

Árvores binárias

- Função que imprime a árvore:

```
void imprimeArvore(Arvore* a) {
    if(!estaVazia(a)) {
        printf("%c ", a->info);
        imprimeArvore(a->esq);
        imprimeArvore(a->dir);
    }
}
```

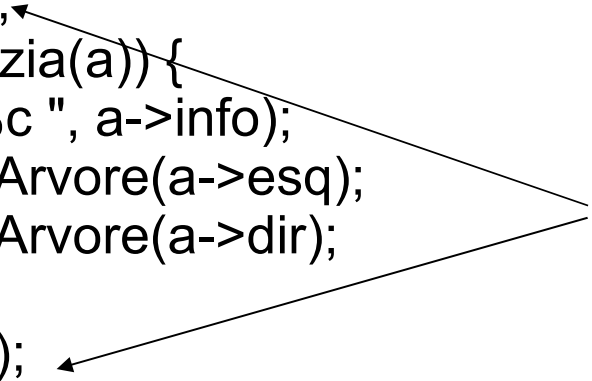
- Como alterar essa função para que ela imprima o conteúdo da árvore na notação textual estudada?

Árvores binárias

Função que imprime a árvore:

```
void imprimeArvore(Arvore* a) {
  printf("<");
  if(!estaVazia(a)) {
    printf("%c ", a->info);
    imprimeArvore(a->esq);
    imprimeArvore(a->dir);
  }
  printf(" >");

}
```



Imprime usando a notação textual apresentada.

Como imprimir a árvore sem recursão???

Árvores binárias

- Para imprimir a árvore sem recursão podemos usar uma pilha como auxiliar:

```
void imprimeArvore(Arvore* arv){
    Arvore* aux;
    Pilha* p = criarPilha();
    push(p, arv);
    while(!eVazia(p)){
            aux = pop(p);
            printf("%c ", aux->info);
            if(aux->dir != NULL)
                    push(p, aux->dir);
            if(aux->esq != NULL)
                     push(p, aux->esq);
    }
}
```

- Árvores binárias
  - Função para liberar espaço alocado para a árvore:

```
Arvore* liberarArvore(Arvore* a){
  if(!estaVazia(a)) {
    liberarArvore(a->esq);
    liberarArvore(a->dir);
    free(a);
  }
  return NULL;
}
```

Libera as subárvores antes de liberar o nó raiz

Retorna árvore atualizada (vazia). Senão não podemos remover nós usando essa função!

Árvores binárias

- Como a definição de árvore é recursiva, ela não faz distinção entre árvores e subárvores.
- Assim, as funções criarArvore e liberarArvore podem ser usadas para inserir uma subárvore em um ramo de uma árvore e liberar (remover) um ramo de uma árvore dada.
  - Ex: a->dir->esq = libera(a->dir->esq);

Árvores binárias

- Função que verifica se um caractere pertence à árvore:

```
int pertenceArvore(Arvore* a, char c){
  if(estaVazia(a)) {
    return 0;
  } else {
    return a->info==c || pertenceArvore(a->esq, c) ||
    pertenceArvore(a->dir, c);
  }
}
```

Interrompe a busca tão logo o elemento seja encontrado.

ESTRUTURAS DE DADOS

Árvores binárias

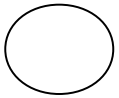
- Ordens de percurso em árvores binárias (métodos de caminhamento): Ordem na qual os nós da árvore serão visitados (para executar alguma operação com a informação).
  - Pré-ordem: visita a raiz, percorre SAE, percorre SAD.
  - Ordem simétrica (In-ordem): percorre SAE, visita raiz, percorre SAD.
  - Pós-ordem: percorre SAE, percorre SAD, visita raiz.

Dependendo da aplicação, uma dessas ordens pode ser preferível, ou seja, a ordem de percurso é importante. Como implementá-las na impressão?

ESTRUTURAS DE DADOS

- Altura de uma árvore

- A **altura da árvore (h)** é o comprimento do caminho mais longo da raiz até uma das folhas.

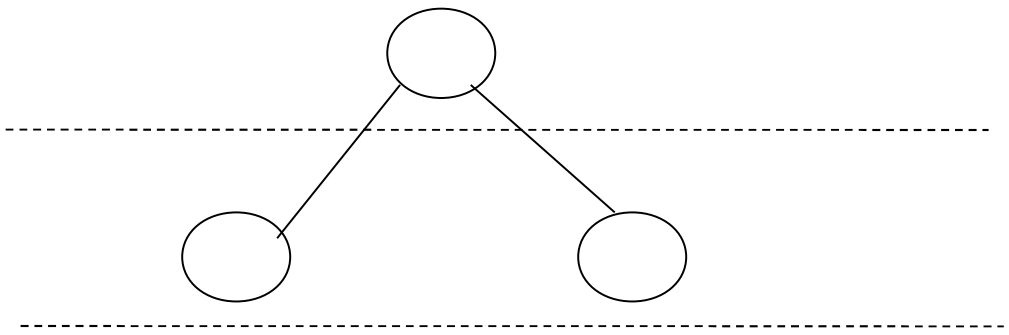


Somente nó raiz: h = 0

ESTRUTURAS DE DADOS

- Altura de uma árvore

- A **altura da árvore (h)** é o comprimento do caminho mais longo da raiz até uma das folhas.

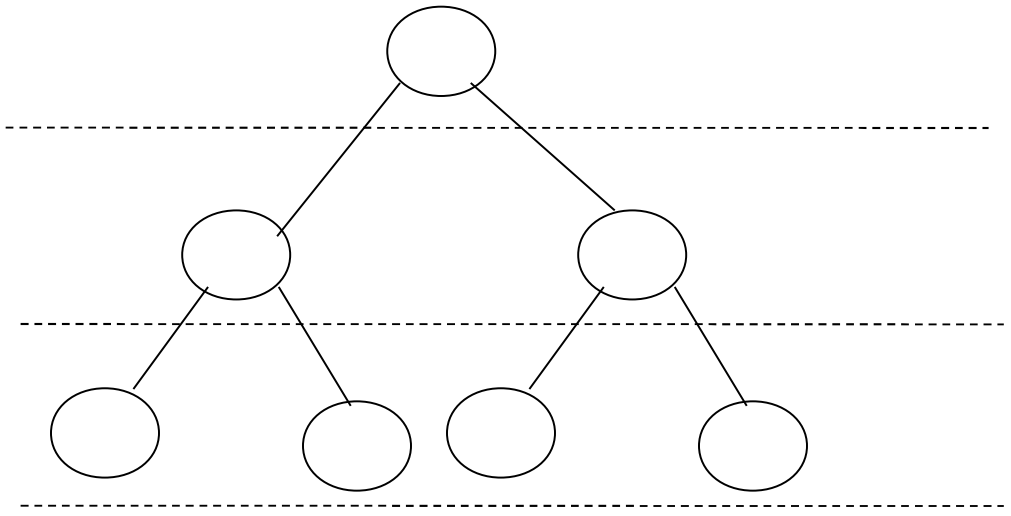


h = 1

ESTRUTURAS DE DADOS

- Altura de uma árvore

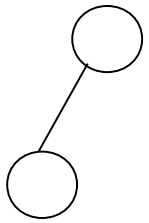
- A **altura da árvore** é o comprimento do caminho mais longo da raiz até uma das folhas.



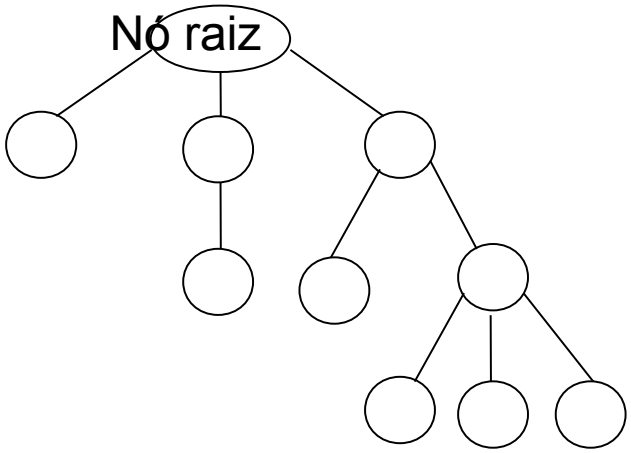
h = 2

Árvores

- **Altura** de uma árvore: comprimento do caminho mais longo da raiz até uma das folhas.



Altura = 1



Altura = 3

Árvores

- A **altura** de uma árvore com um único nó raiz é 0 (zero).

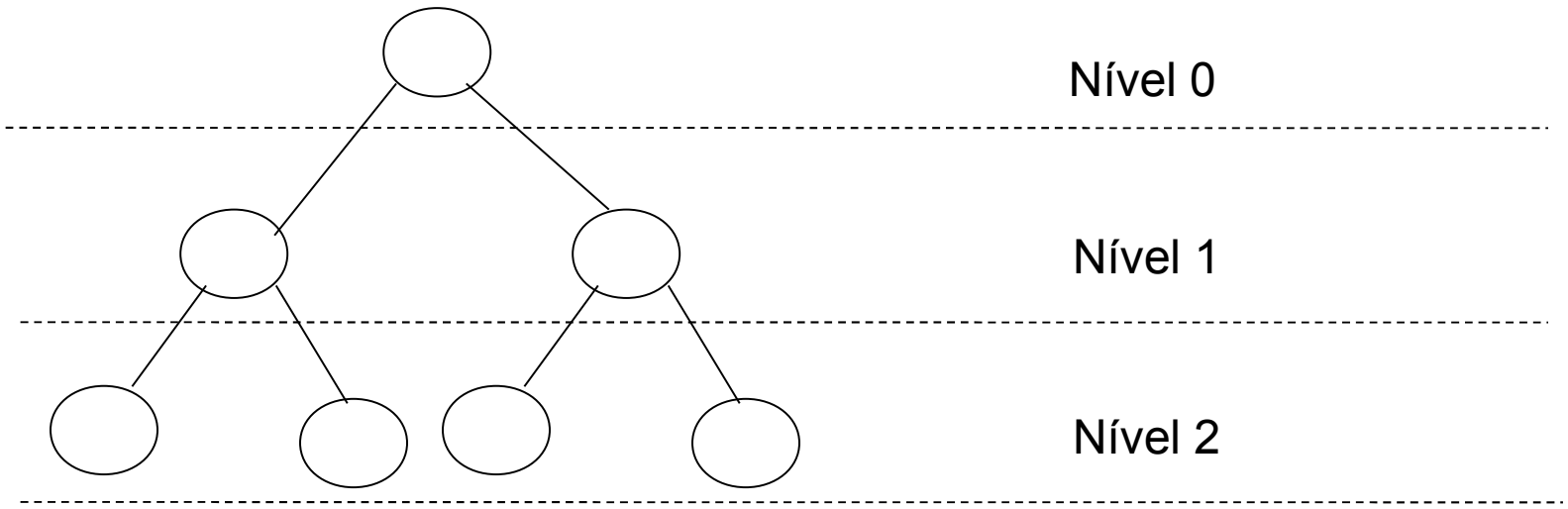
Altura = 0

- A **altura** de uma árvore vazia é negativa e vale -1.

ESTRUTURAS DE DADOS

- Altura de uma árvore

- Também podemos enumerar os níveis em que os nós aparecem na árvore:

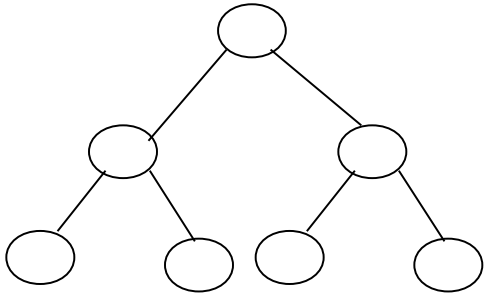


E assim por diante. O último nível da árvore é o nível h, sendo h a altura da árvore.

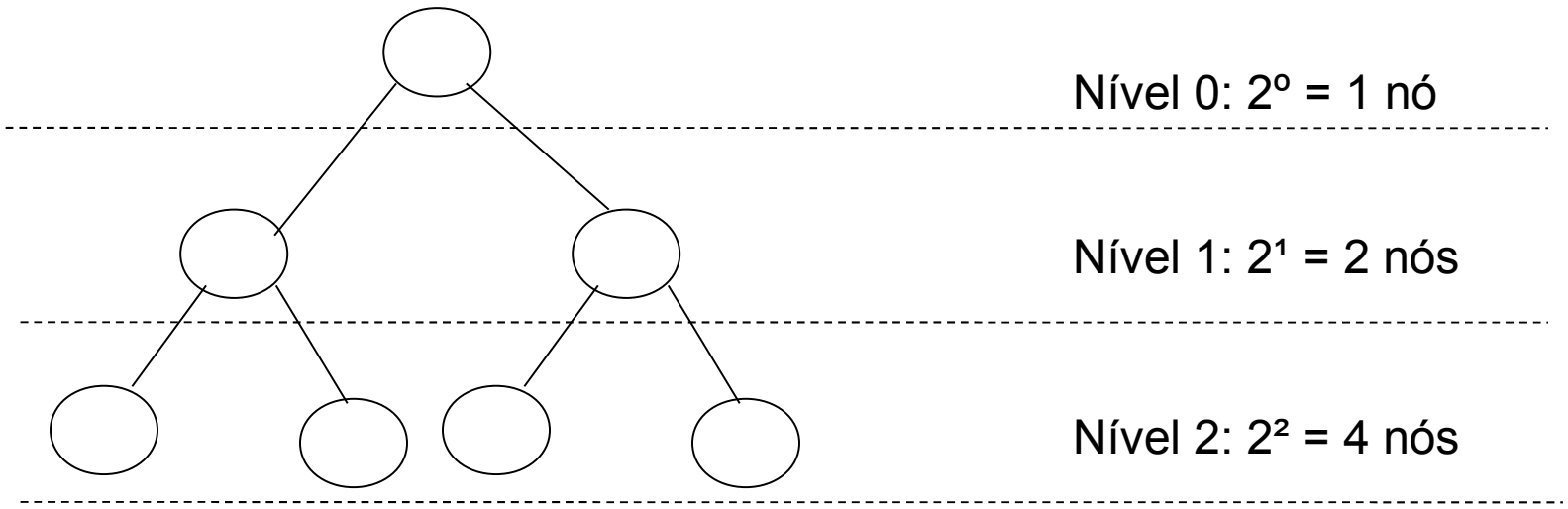
Árvores binárias

- Árvore binária **cheia (completa):**
  - todos os nós internos, ou seja, exceto as folhas, têm 2 descendentes (2 subárvores associadas).
  - todos os nós folhas estão no último nível.

- Exemplo:



- Árvore binária:
  - Quantidade de nós em uma árvore cheia de altura h:



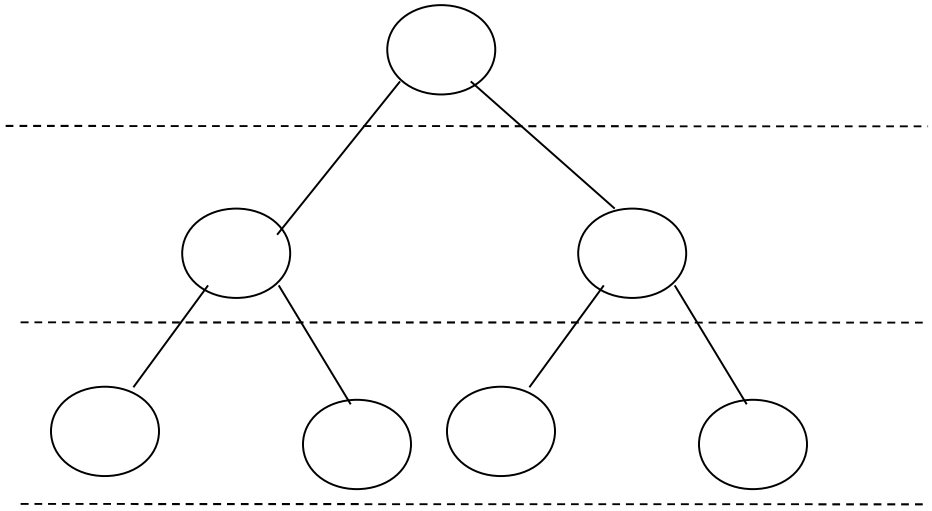
E assim por diante: No nível n temos 2^n nós.

- Árvore binária:
  - **Relação entre o número de nós de uma árvore binária e sua altura**: A cada nível, o número de nós vai dobrando, de maneira que uma árvore cheia de altura h pode ter um número de nós dado por:

$$2^0 + 2^1 + 2^2 + \ldots + 2^{h-1} + 2^h = \sum_{i=0}^{h} 2^i$$

$$\sum_{i=0}^{h} 2^i = 2^{h+1} - 1$$

- Árvore binária:
  - Observe que o número de nós em um nível n é uma unidade a mais do que a soma de todos os nós dos níveis anteriores:



$$2^n = 1 + \sum_{i=0}^{n-1} 2^i$$

- Árvore binária:
  - Observe que o número de nós em um nível n é uma unidade a mais do que a soma de todos os nós dos níveis anteriores:

$$2^n = 1 + \sum_{i=0}^{n-1} 2^i$$

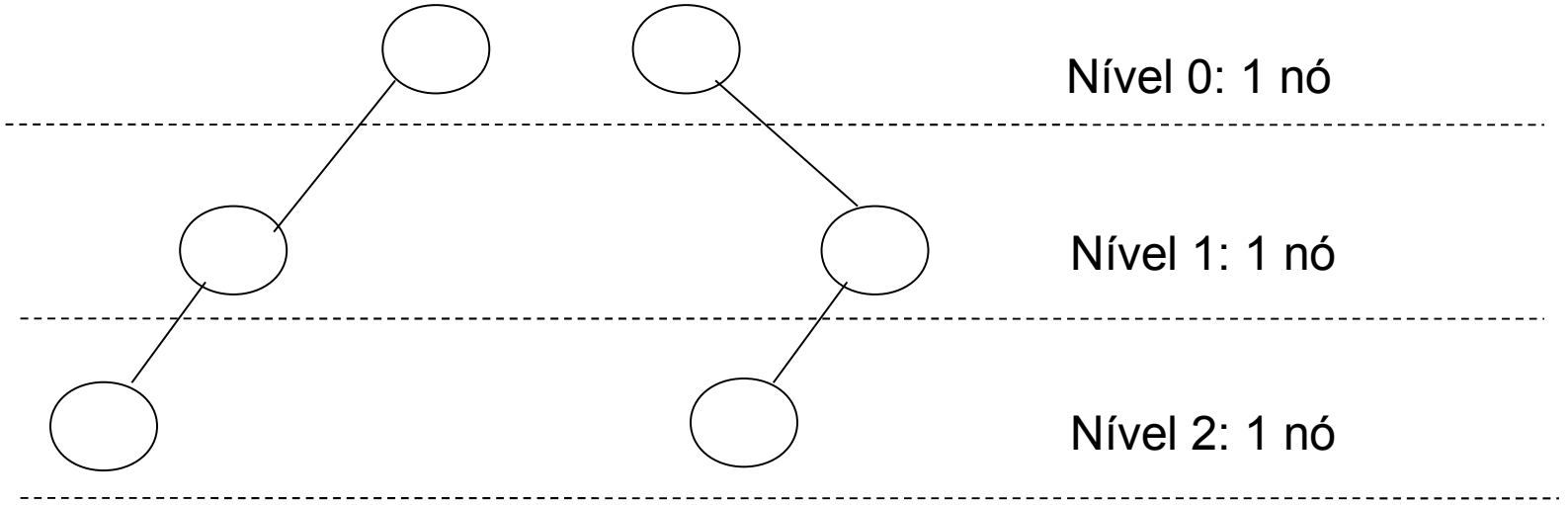
  - Então, confirmamos que uma árvore cheia de altura h tem um número de nós dado por:

$$qtde = 2^{h+1} - 1$$

Número de nós do próximo nível

Árvores binárias

- Árvore degenerada: todos os seus nós internos, têm 1 único descendente (uma única subárvore associada), com exceção da (única) folha.



Uma árvore degenerada de altura h tem h + 1 nós.