Árvores

Disciplina: Estrutura de Dados

Luciano Moraes Da Luz Brum

Universidade Federal do Pampa – Unipampa – Campus Bagé

Email: lucianobrum18@gmail.com

Universidade Federal do Pampa

Tópicos

- Conceitos básicos.
  - Conceitos.
  - Formas de Representação.
  - Altura de uma árvore.
  - Árvore cheia e degenerada.
  - Grau de uma árvore.
- Árvores Binárias.
- Representação em C.
- Interface do tipo “arvore”.
- Árvores com número variável de filhos.
- Resumo.

Conceitos Básicos

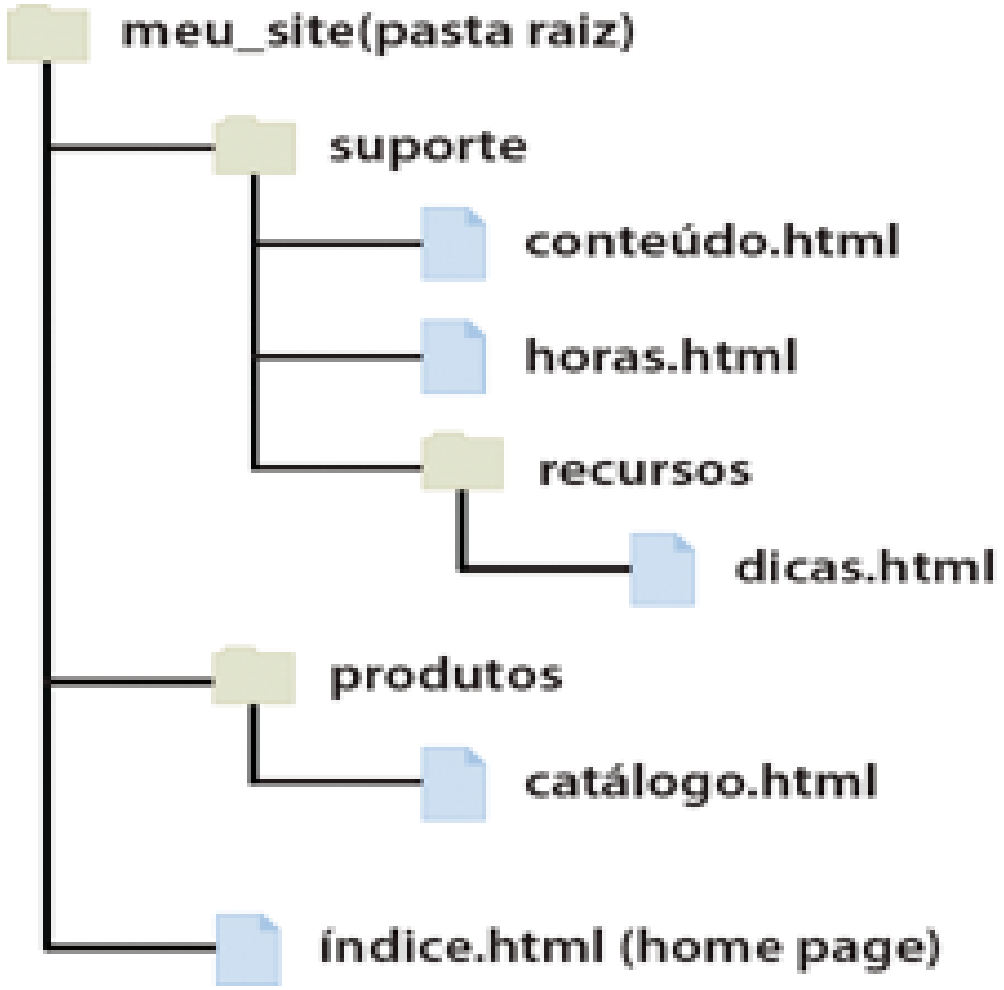
- Até o momento vimos estruturas de dados lineares, como filas, listas e pilhas.

- É inegável a sua importância, porém, não são adequadas para dados que devem ser dispostos de forma hierárquica.

Conceitos Básicos

- Exemplos:
  - Quando são criados arquivos no computador, eles são armazenados em uma estrutura hierárquica de diretórios (pastas).
  - Há um diretório base, onde podemos armazenar vários arquivos e subdiretórios. Dentro dos subdiretórios, isso também é possível. E assim por diante, recursivamente.

Conceitos Básicos



Conceitos Básicos

- Exemplos:
  - Árvores genealógicas.
  - Estrutura hierárquica de uma organização.
  - Sites de Internet.

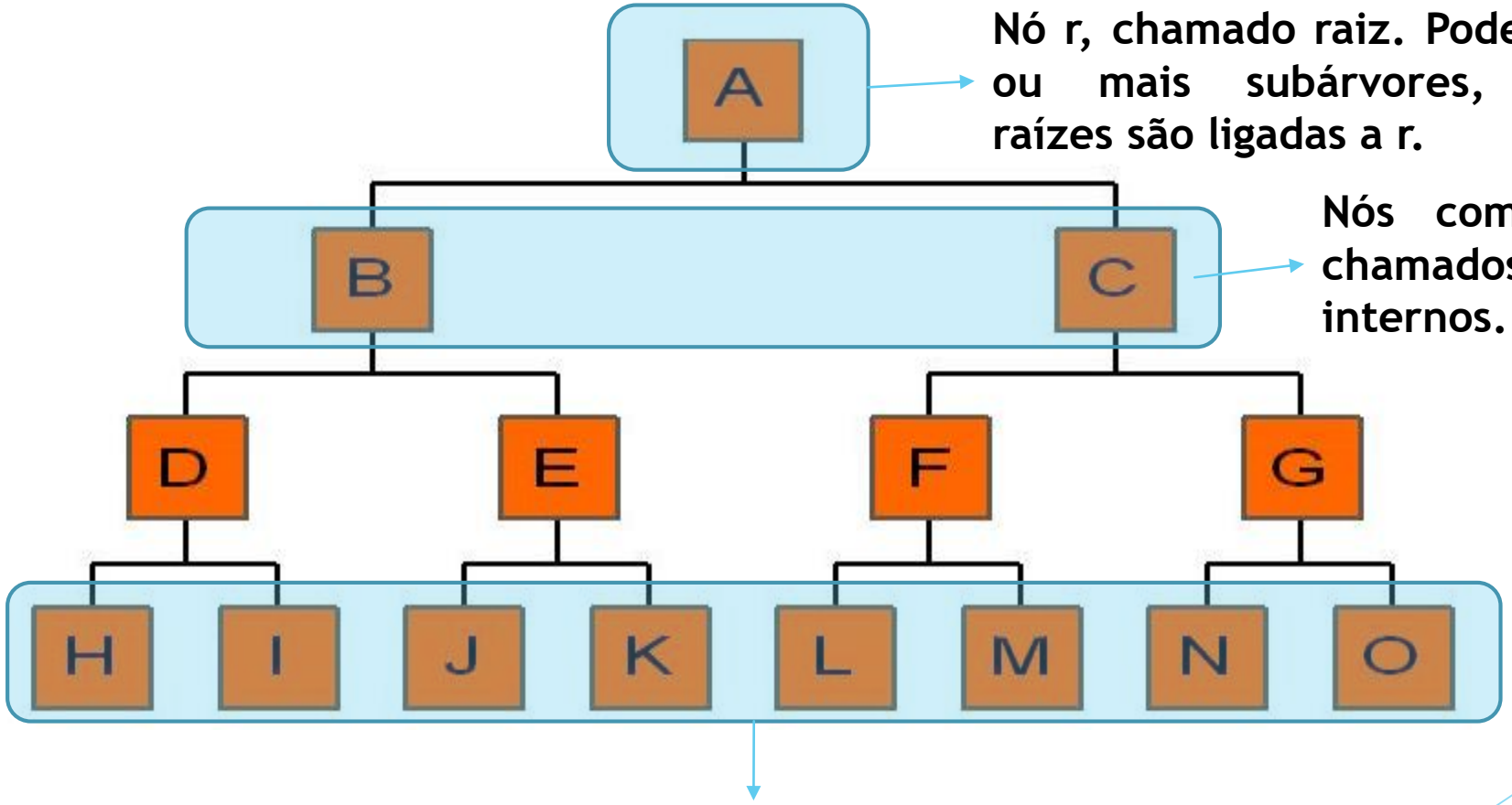
Conceitos Básicos

- Estruturas de dados adequadas para representar dados de forma hierárquica são as árvores.

- A forma mais natural de definir estrutura de árvore é através da recursividade.

- Como representamos visualmente uma árvore?

Conceitos Básicos



Nó r, chamado raiz. Pode ter 0 ou mais subárvores, cujas raízes são ligadas a r.

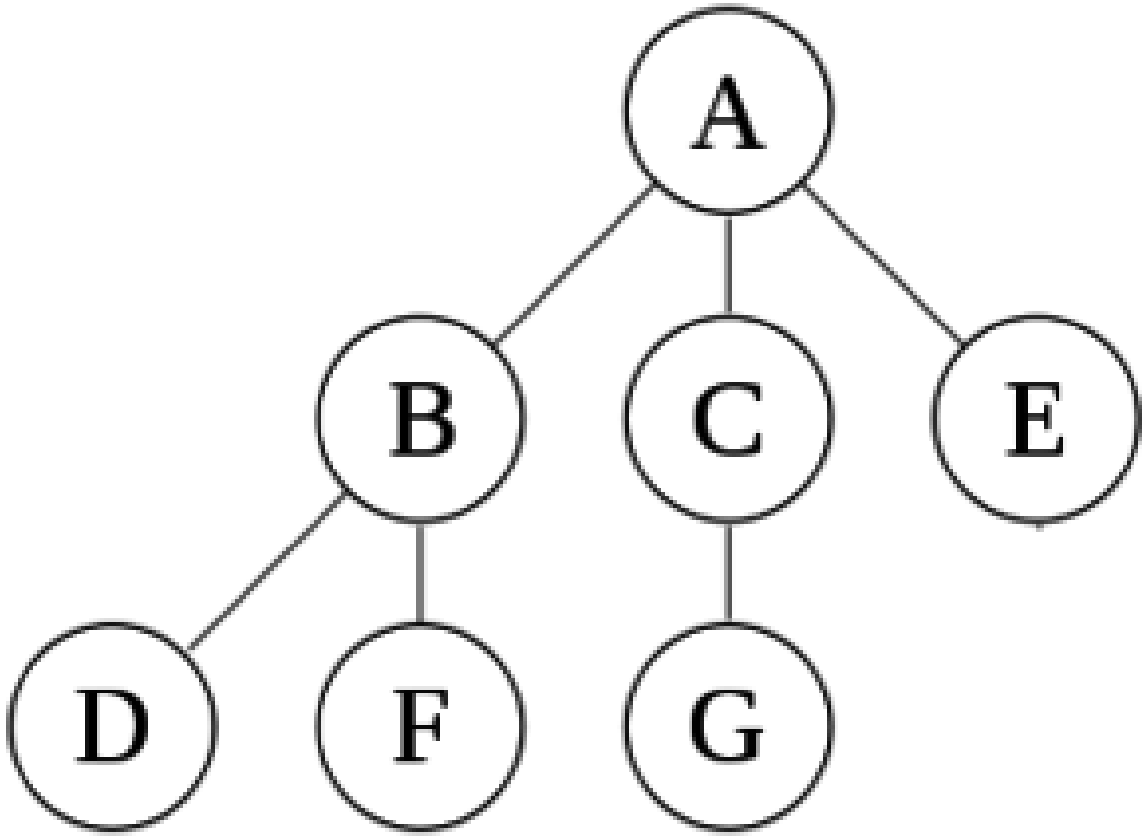
Nós com filhos, chamados nós internos.

Nós que não tem filhos são chamados de nós folhas.

Formas de Representação

- Temos as seguintes formas de representação de uma árvore:
  - Hierárquica.
  - Diagrama de Inclusão.
  - Diagrama de Barras.
  - Parênteses aninhados.

Formas de Representação



Hierarquica

Formas de Representação

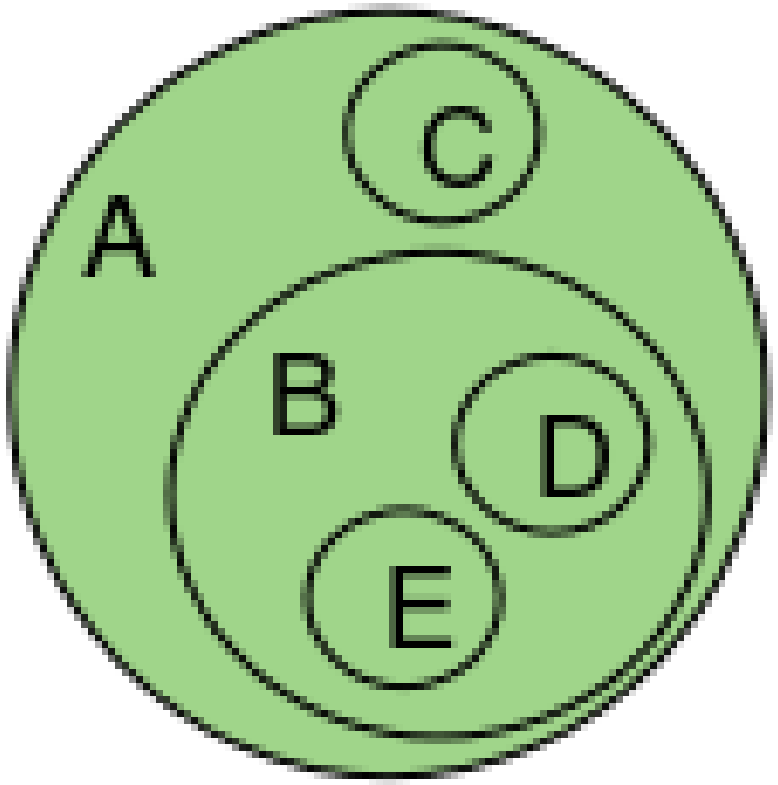


Diagrama de Inclusão

Formas de Representação

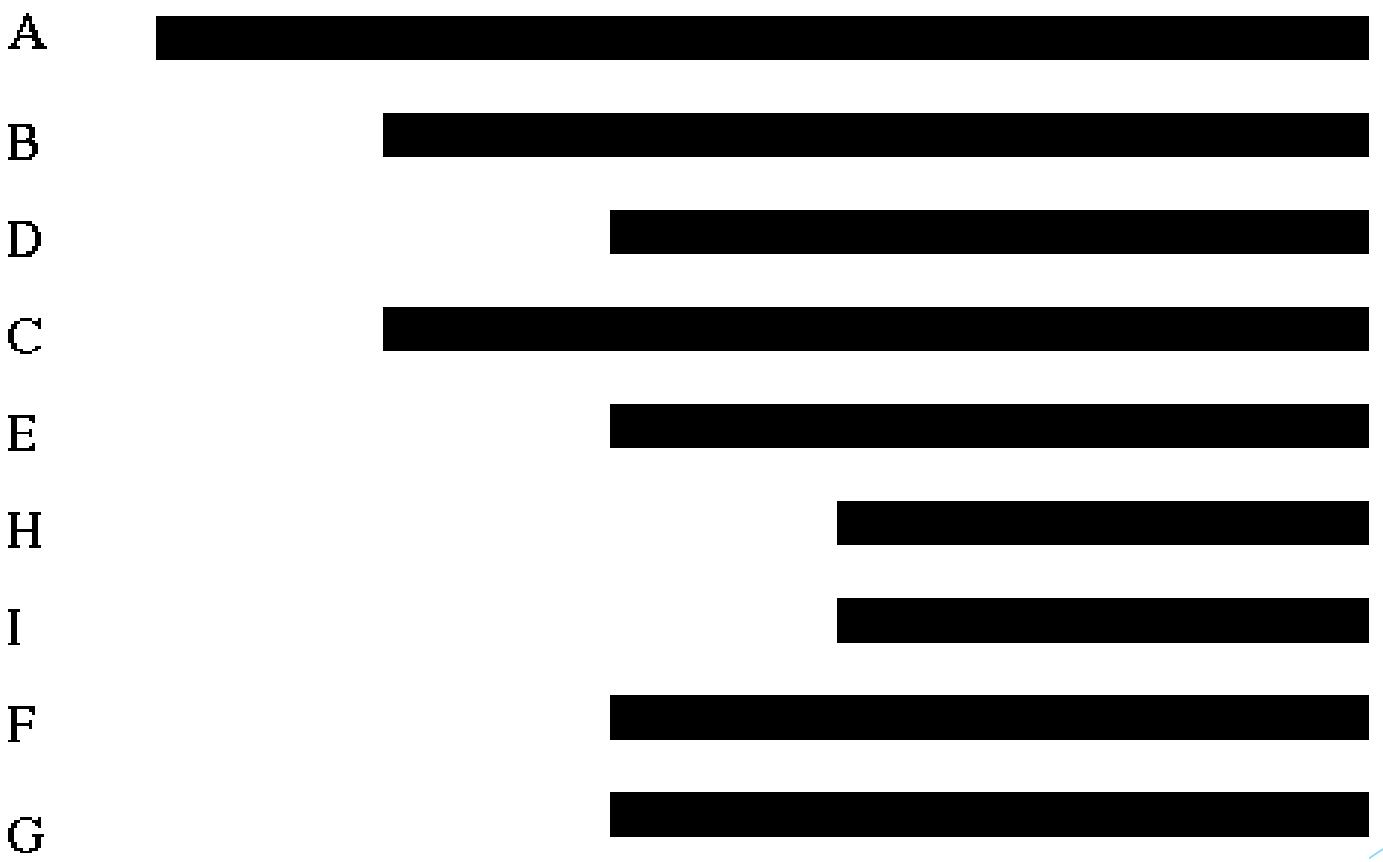
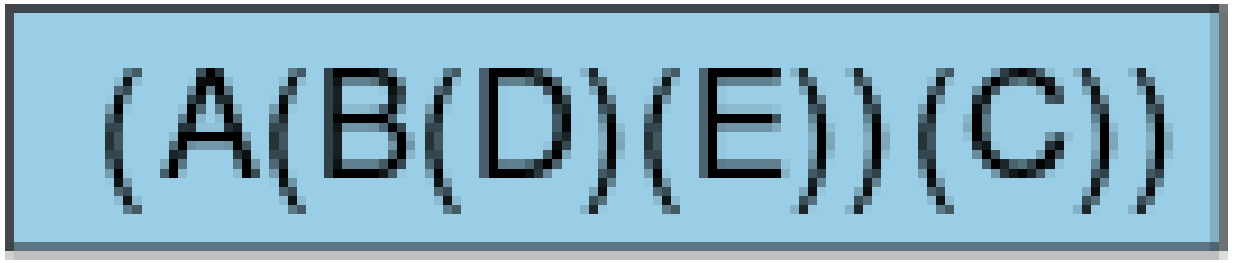


Diagrama de Barras

Formas de Representação



Parênteses Aninhados

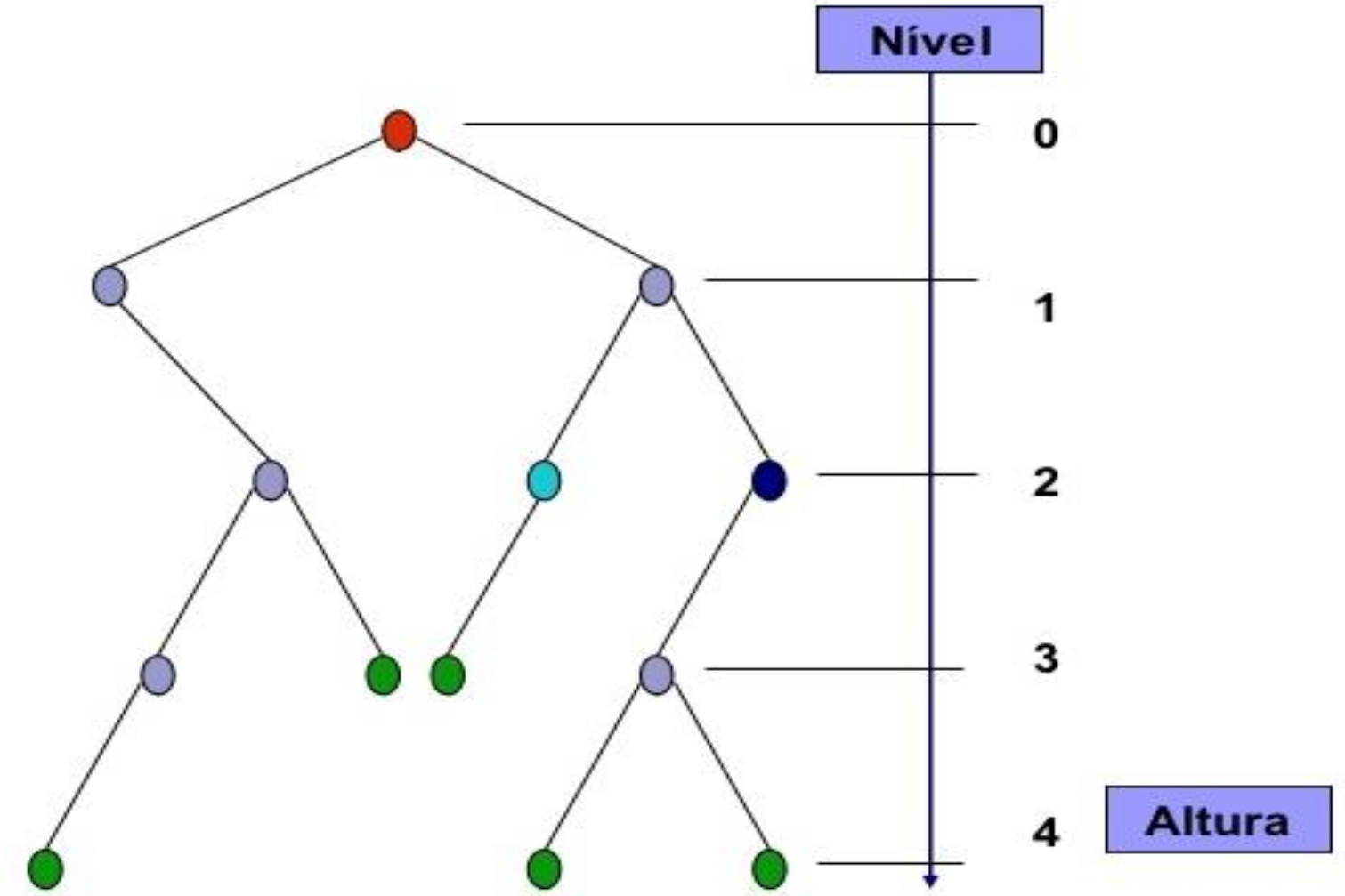
Altura de uma árvore

- Uma propriedade fundamental de todas as árvores é que:
  - Só existe um caminho da raiz para qualquer nó.
- Podemos definir a altura da árvore como sendo o comprimento de caminho mais longo da raiz até uma das folhas.
- Comprimento do caminho: um caminho de k vértices é obtido pela sequencia de k-1 pares. K-1 é o comprimento do caminho.

Altura de uma árvore

- Altura de árvores com apenas um nó raiz é zero.
- Altura de uma árvore vazia vale -1, como valor padronizado.
- Podemos numerar os níveis em que os nós aparecem numa árvore, sendo:
  - Nó raiz o nível 0.
  - O último nó da árvore o nível ‘h’, sendo ‘h’ a altura da árvore.

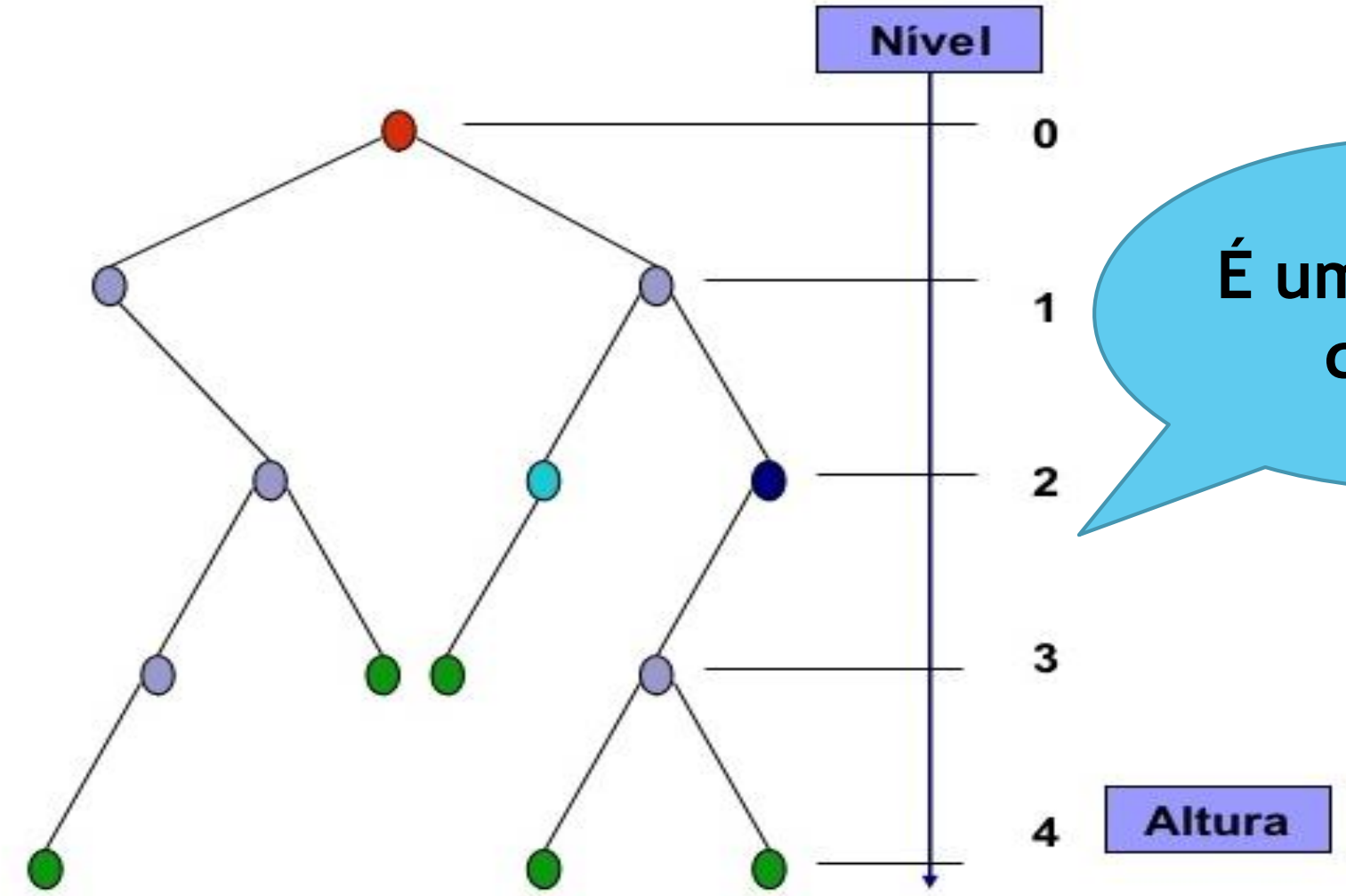
Altura de uma árvore



Árvores Cheias e Degeneradas

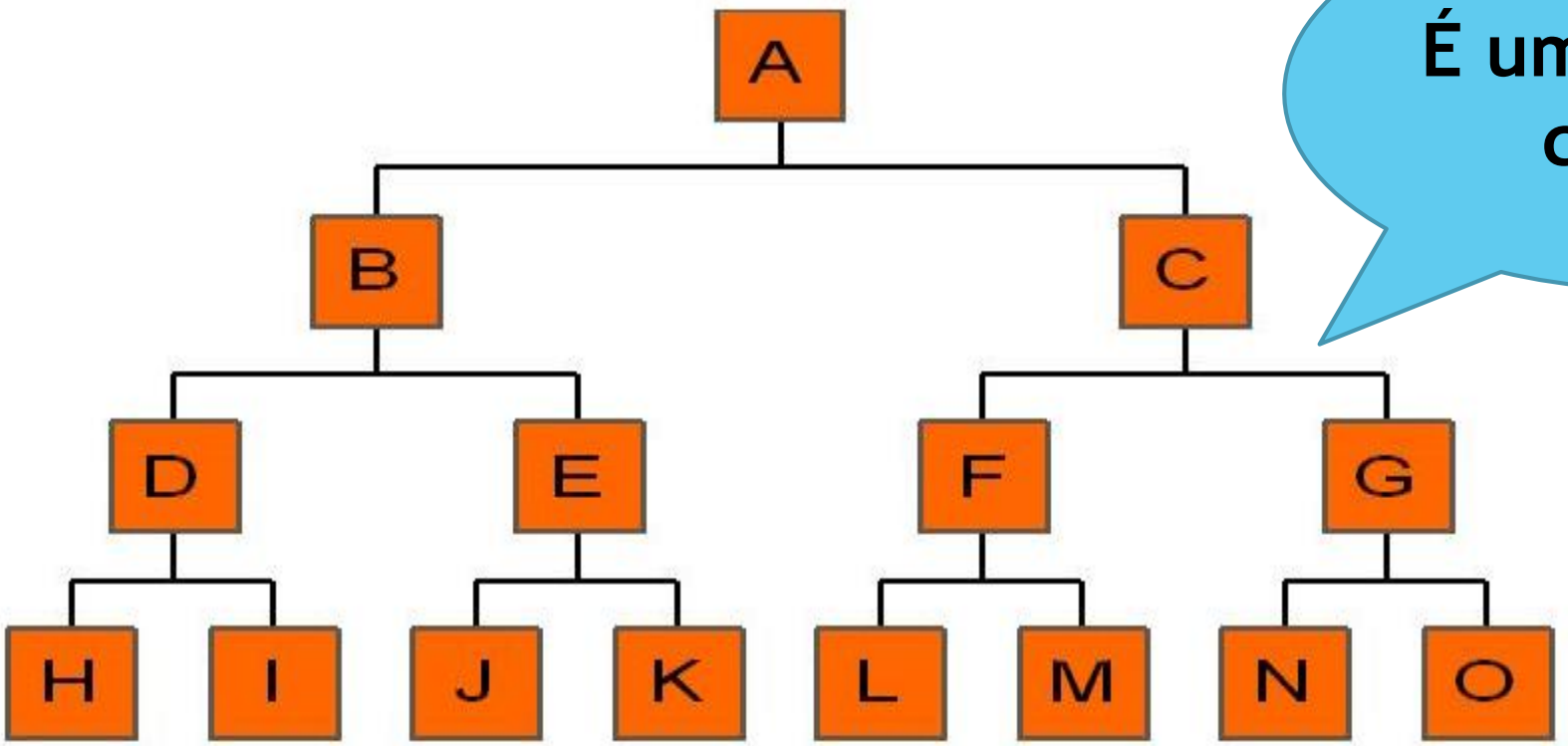
- Podemos ainda ter árvores cheias e degeneradas:
  - Uma árvore é dita cheia (ou completa), se:
    - Todos os nós internos têm o máximo de subárvores associadas.
    - Todos os nós folhas estão no último nível.

Árvores Cheias e Degeneradas



É uma árvore cheia?

Árvores Cheias e Degeneradas



É uma árvore cheia?

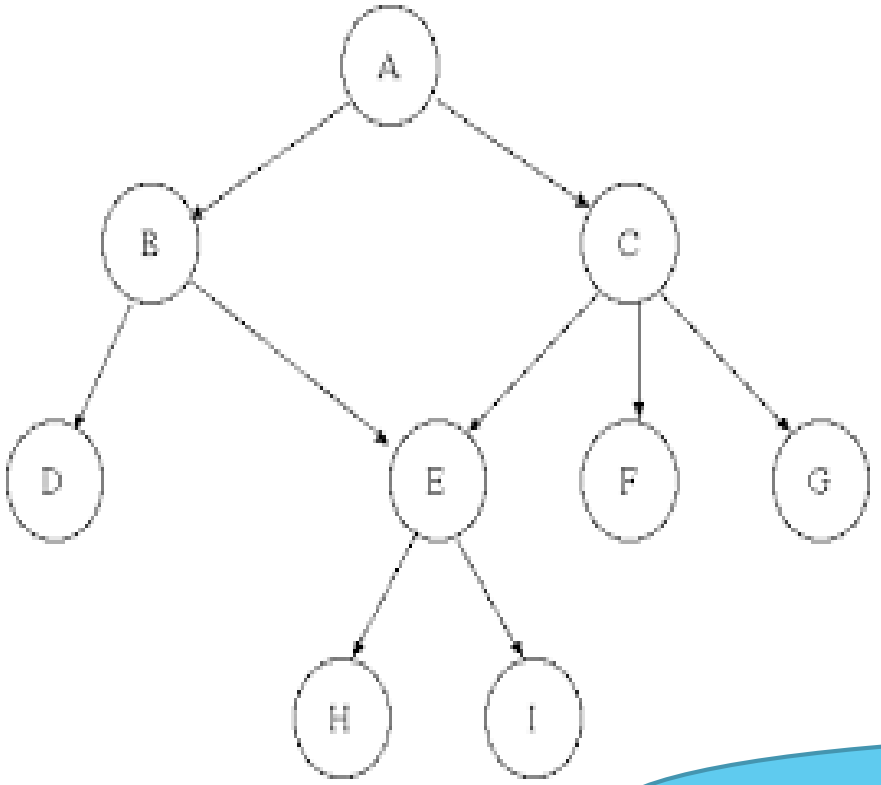
Árvores Cheias e Degeneradas

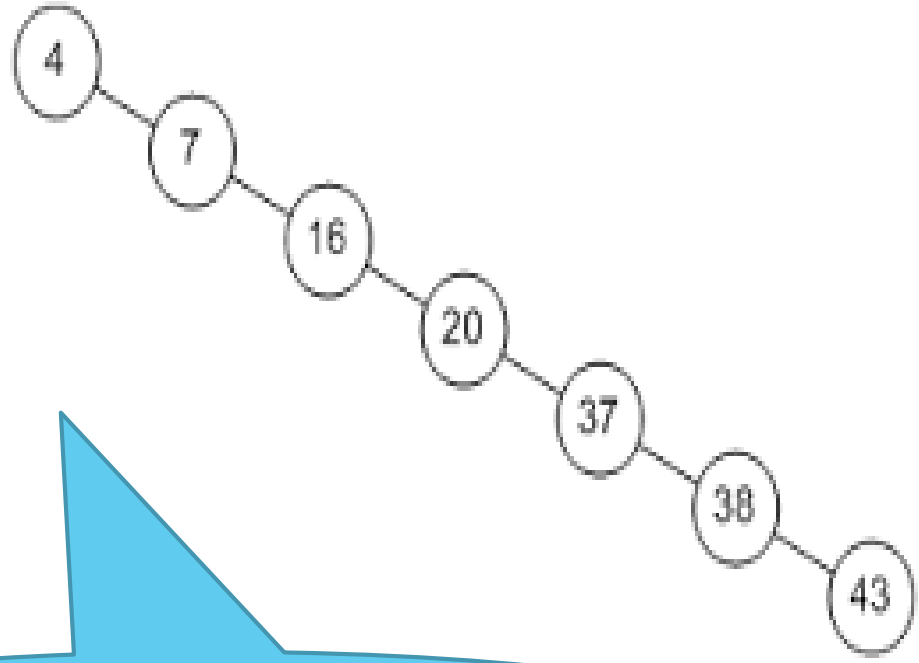
- Uma árvore é dita degenerada, se todos os seus nós internos têm uma única subárvore associada.

- A estrutura de árvore degenera para uma estrutura linear.

- Como consequência, temos apenas um nó por nível, sendo que uma árvore degenerada de altura ‘h’ tem ‘h+1’ nós.

Árvores Cheias e Degeneradas





Qual é degenerada?

Grau de uma Árvore

- O grau de uma árvore é definido pelo máximo grau de saída de seus nós.

- O grau de saída de um nó é a quantidade máxima de filhos por nó.

- Qual o grau de um nó interior?

Conceitos Básicos

- O número de filhos permitido por nó e as informações armazenadas em cada nó diferenciam os vários tipos de árvores existentes.

- Veremos nesta disciplina:

  - Árvores binárias (árvores de grau 2, ou seja, cada nó tem no máximo, dois filhos).

  - Conceitualmente, árvores com n° variável de filhos (grau n).

Tópicos

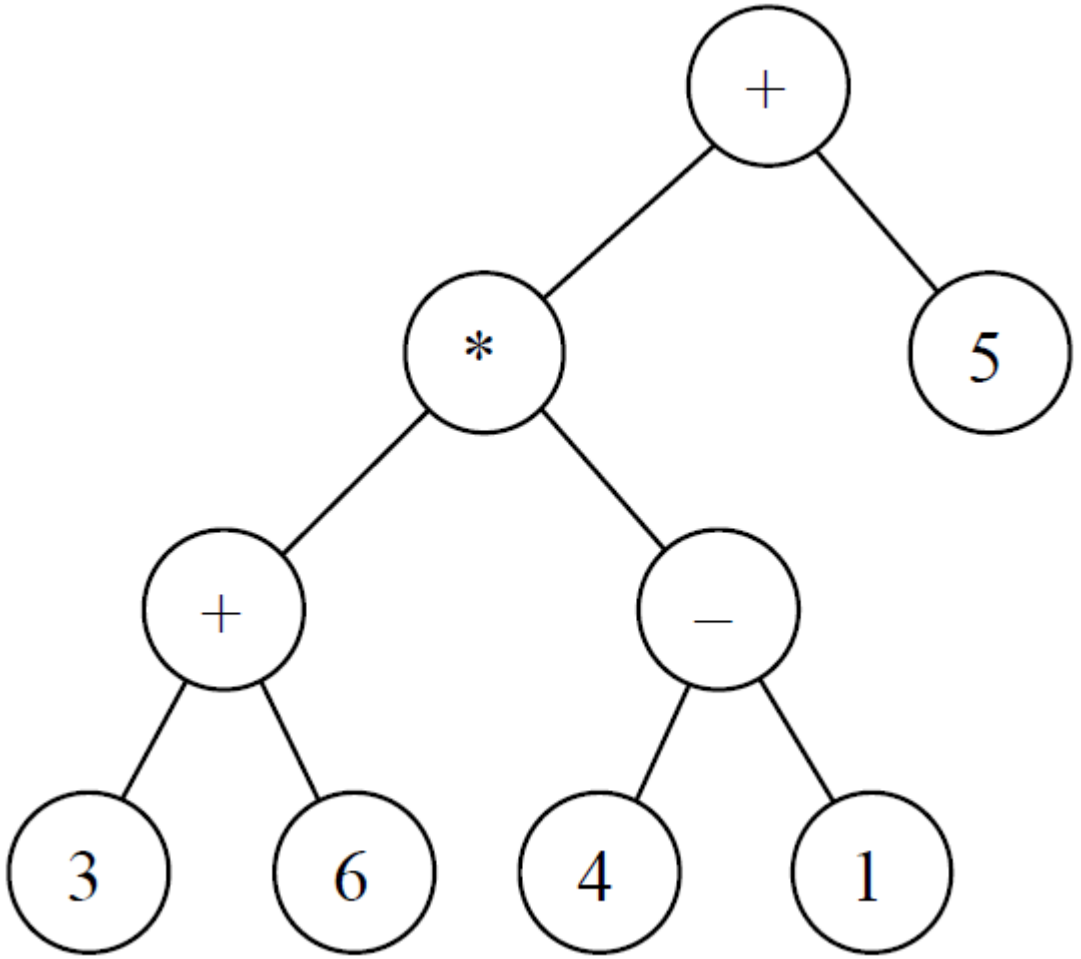
- ~~Conceitos básicos.~~
- Árvores Binárias.
  - Conceitos.
  - Ordens de percurso.
  - Altura de uma árvore binária.
- Representação em C.
- Interface do tipo “arvore”.
- Árvores com número variável de filhos.
- Resumo.

Árvores Binárias

- Uma árvore binária é aquela que possui grau 2, ou seja, possui um número máximo de 2 filhos por nó.

- Um exemplo de utilização é a avaliação de expressões, como mostra a figura a seguir.

Árvores Binárias



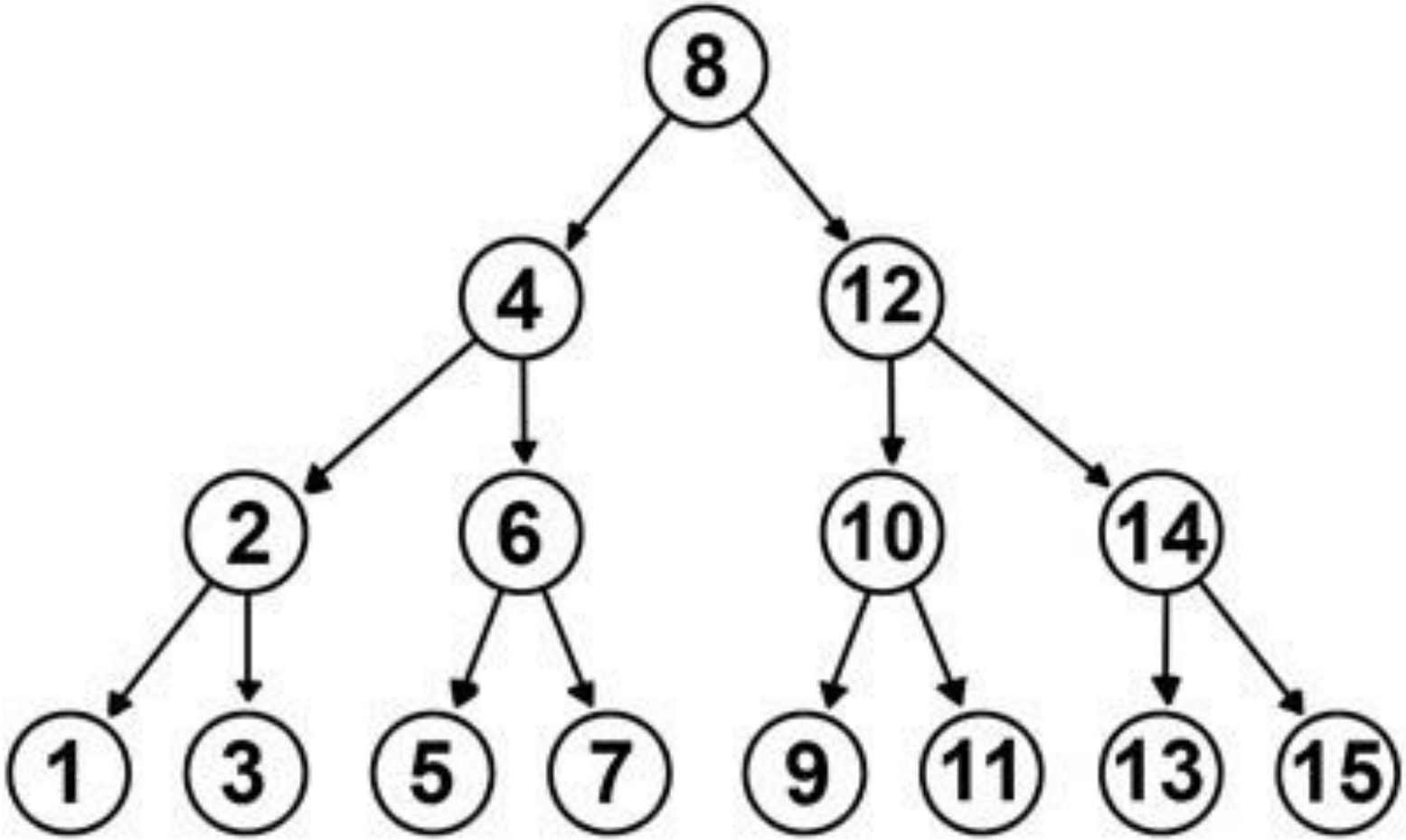
Á árvore ao lado representa a expressão: (3+6)*(4-1)+5

Árvores Binárias

- Em uma árvore binária, um nó pode ter zero, um ou dois filhos. Podemos definir recursivamente como sendo:

  - **Uma árvore vazia; ou**
  - **Um nó raiz tendo 2 subárvores, identificadas como subárvore direita (*sad*) e subárvore esquerda (*sae*).**

Árvores Binárias

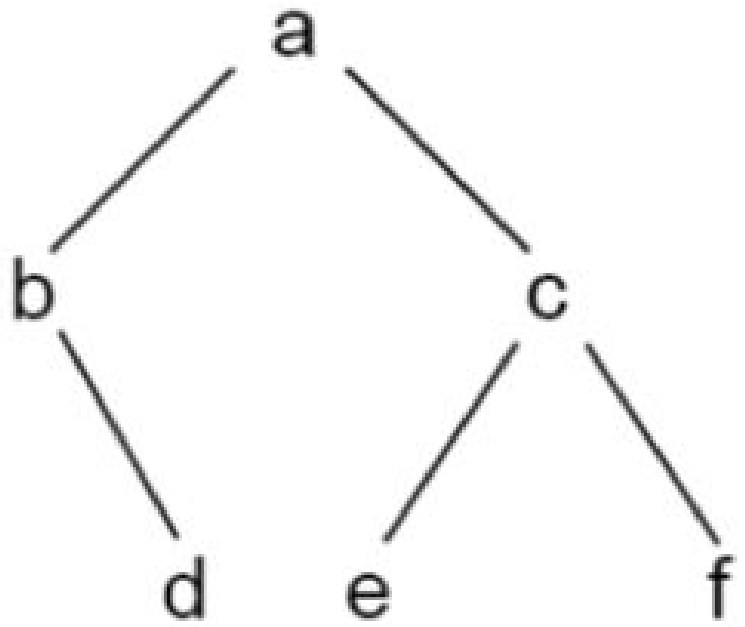


Árvores Binárias

- Para descrever árvores binárias, podemos utilizar a seguinte notação textual:
  - **Árvore vazia: < >**
  - **Árvore não-vazia: <raiz sad sae>**

Árvores Binárias

- Como representar a árvore a seguir?



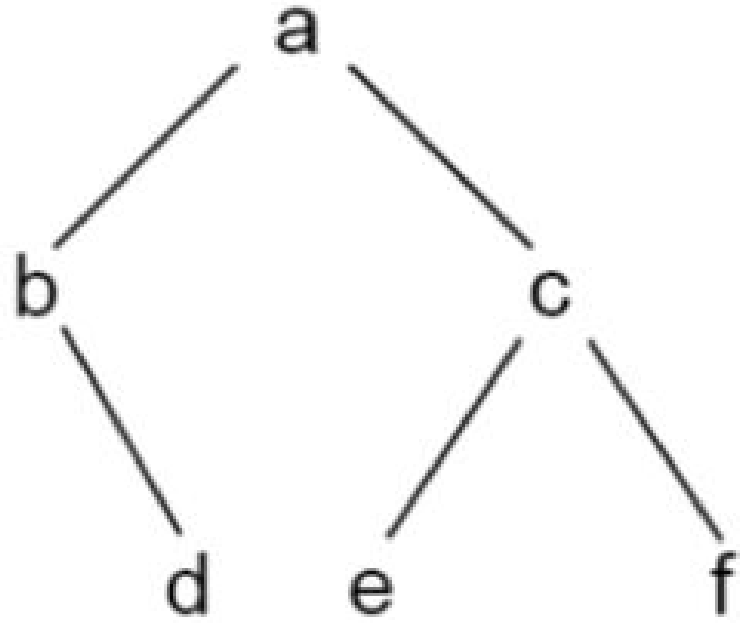
<a <b<><d<><><>> <c<e<><><>><f<><><>> >

Ordem de Percurso

- Existem 3 formas de percorrer os elementos de uma árvore:
  - **Pré-ordem: raiz -> sae -> sad**
  - **Ordem simétrica: sae -> raiz -> sad**
  - **Pós-ordem: sae -> sad -> raiz**

Ordem de Percurso

- Como representar a árvore a seguir nas 3 ordens de percurso?



Pré-ordem:<a<b<>><d<><>>><c<e<><>><f<><>>>>

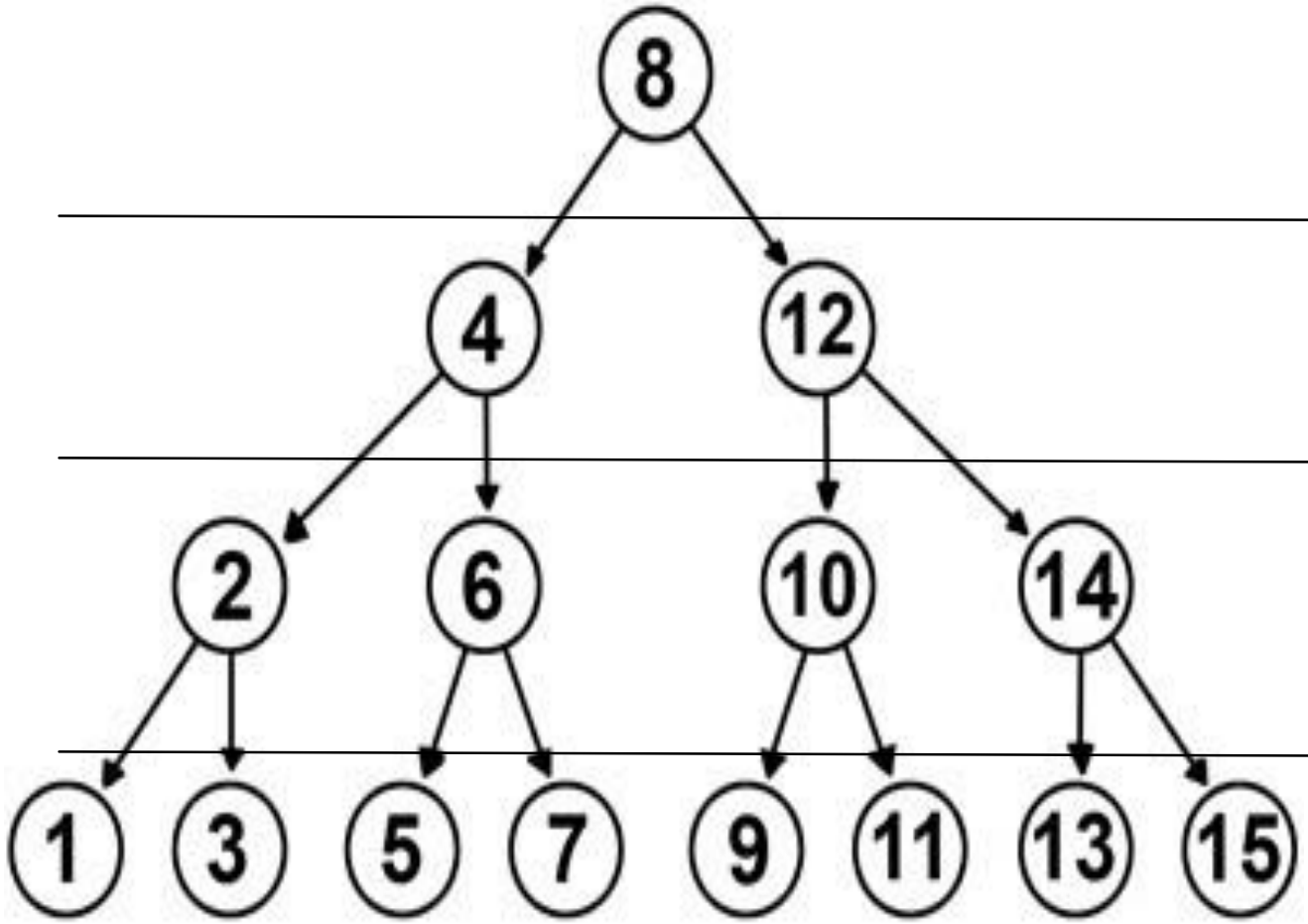
Simétrico:<<<>b<<>d<>>>a<<<>e<>>c<<>f<>>>>

Pós-ordem: <<<><<><><>d>b><<<><>e><<><>f>c>a>

Altura da Árvore Binária

- Se temos uma árvore binária cheia, podemos perceber que:
  - **No nível 0 temos 1 nó;**
  - **No nível 1 temos 2 nós;**
  - **No nível 2 temos 4 nós.**
  - **No nível 3 temos 8 nós.**
  - **No nível ‘n’ temos 2^n nós.**

Árvores Binárias



Nível 0: 2^0 = 1 nó

Nível 1: 2^1 = 2 nós

Nível 2: 2^2 = 4 nós

Nível 3: 2^3 = 8 nós

Altura da Árvore Binária

- Pergunta: é correto afirmarmos que o número de nós de determinado nível é igual a soma de todos os nós dos níveis anteriores mais 1?

- Sim, portanto é possível mostrar que uma árvore cheia de altura h tem um número de nós dado por: $2^{h+1} - 1$

unipampa
Universidade Federal do Pampa

Tópicos

- ~~Conceitos básicos.~~
- ~~Árvores Binárias.~~
- **Representação em C.**
- **Interface do tipo “arvore”.**
- **Árvores com número variável de filhos.**
- **Resumo.**

Representação em C

- Podemos definir um tipo para representar uma árvore binária.

- **Cada nó deve armazenar três informações:**
  - **A informação propriamente dita;**
  - **Um ponteiro para a subárvore esquerda;**
  - **Um ponteiro para a subárvore direita;**

Representação em C

```
struct arv {
    char info;
    struct arv *esq;
    struct arv *dir;
};
Typedef struct arv Arv;
```

Representação em C

- Da mesma forma que a lista encadeada é representada por um ponteiro para o primeiro nó, a estrutura da árvore é representada por um ponteiro para o nó raiz.

- **Dado o ponteiro do nó raiz, podemos ter acesso aos demais nós.**

unipampa
Universidade Federal do Pampa

Tópicos

- ~~Conceitos básicos.~~
- ~~Árvores Binárias.~~
- ~~Representação em C.~~
- **Interface do tipo “arvore”.**
- **Árvores com número variável de filhos.**
- **Resumo.**

Interface do tipo “arvore”

- Como veremos a seguir, as funções que manipulam árvores são implementadas, em geral, de forma recursiva.

- **Veremos as seguintes operações com árvores binárias.**

Interface do tipo “arvore”

- Arv* arv_criavazia();
- Arv* arv_cria(char c, Arv *sae, Arv *sad);
- int arv_vazia(Arv *a);
- void arv_imprime_pre_ordem(Arv *a);
- void arv_imprime_simetrico(Arv *a);
- void arv_imprime_pos_ordem(Arv *a);
- Arv* arv_libera(Arv *a);
- int arv_pertence(Arv *a, char c);
- int arv_altura(Arv *a);

Interface do tipo “arvore”

- Para criar a estrutura da árvore vazia:

```
Arv* arv_criavazia(){
    return NULL;
}
```

Interface do tipo “arvore”

- Para construir arvores não-vazias, podemos ter uma operação que cria um nó raiz com a informação dada e duas subarvores, esquerda e direita. O valor de retorno é o endereço do nó raiz criado.

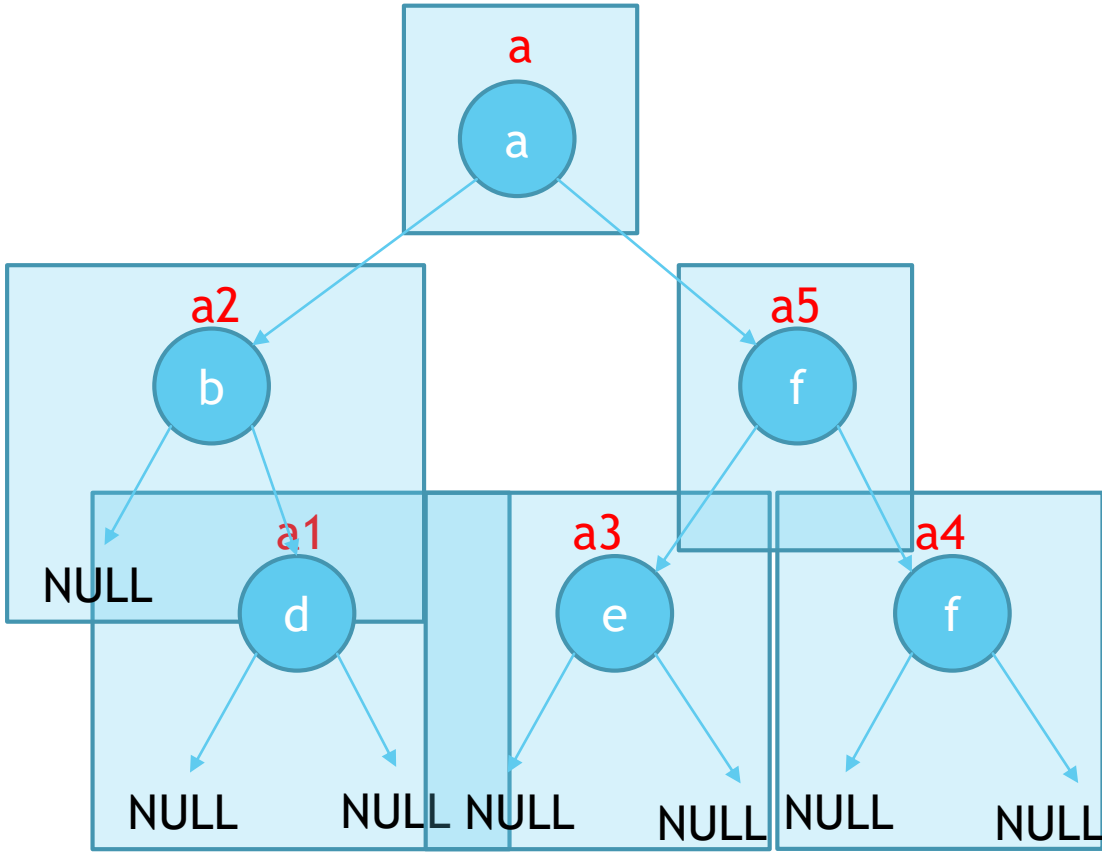
```
Arv* arv_cria(char c, Arv* sae, Arv* sad){
    Arv *a=(Arv*)malloc(sizeof(Arv));
    a->info = c;
    a->esq = sae;
    a->dir = sad;
    return a;
}
```

Interface do tipo “arvore”

- As duas funções anteriores representam os casos de definição recursiva de árvore binária:
  - Uma árvore binária ou é vazia, ou é composta por raiz e outras 2 subarvores.
  - Tendo essas funções, podemos criar árvores mais complexas.

Interface do tipo “arvore”

- Exemplo:
  - Arv *a1 = arv_cria(‘d’,arv_criavazia(),arv_criavazia());
  - Arv *a2 = arv_cria(‘b’,arv_criavazia(),a1);
  - Arv *a3 = arv_cria(‘e’,arv_criavazia(),arv_criavazia());
  - Arv *a4 = arv_cria(‘f’,arv_criavazia(),arv_criavazia());
  - Arv *a5 = arv_cria(‘c’,a3,a4);
  - Arv *a = arv_cria(‘a’,a2,a5);



Interface do tipo “arvore”

- Podemos criar a árvore anterior com uma única atribuição, de forma recursiva:

- Arv *a = arv_cria(‘a’,
  arv_cria(‘b’,
  arv_criavazia(),
  arv_cria(‘d’, arv_criavazia(), arv_criavazia())
  ),
  arv_cria(‘c’,
  arv_cria(‘e’, arv_criavazia(), arv_criavazia()),
  arv_cria(‘f’, arv_criavazia(), arv_criavazia())
)
  );

Interface do tipo “arvore”

- A operação a seguir informa se a árvore é vazia:

```
int arv_vazia(Arv *a){
    return a==NULL;
}
```

Interface do tipo “arvore”

- Podemos ter uma operação que retorna ‘1’ se um dado foi encontrado na árvore ou ‘0’ se determinado dado não foi encontrado na árvore.

```
int arv_pertence(Arv *a, char c){
    if(arv_vazia(a)){
        return 0;
    }
    else{
    return a->info == c || arv_pertence(a->esq,c) || arv_pertence(a->dir,c);
    }
}
```

Interface do tipo “arvore”

- Outra função útil consiste em mostrar as informações da árvore.

- A função deve percorrer recursivamente a árvore, visitando todos nós e mostrando sua informação.

- Podemos usar a definição recursiva de árvore. Primeiro testamos se ela é vazia, se não for, imprime conteúdo da raiz e chamamos (recursivamente) a mesma função para imprimir as subarvores.

Interface do tipo “arvore”

```
void arv_imprime(Arv *a){
    printf(“<“);
if(!arv_vazia(a)){
    if(!arv_vazia(a)){
    printf(“%c”,a->info);
        printf(“%c”,a->info);
    arv_imprime(a->esq);
        arv_imprime(a->esq);
    arv_imprime(a->dir);
        arv_imprime(a->dir);
    }
    printf(“>”);
    }
}
}
```

Obs: podemos usar essa função para também mostrar a estrutura da árvore, por meio da notação textual mostrada anteriormente.

Interface do tipo “arvore”

```
void arv_imprime(Arv *a){
if(!arv_vazia(a)){
    arv_imprime(a->esq);
    arv_imprime(a->dir);
    printf("%c", a->info);
    }
}
```

Obs: Podemos alterar a ordem de percurso da árvore modificando a ordem da chama das funções arv_imprime, conforme segue.

Interface do tipo “arvore”

- A altura de uma árvore é uma medida importante na avaliação da eficiência com que visitamos nós de uma árvore;

- Uma árvore binária com ‘n’ nós tem uma altura mínima proporcional a log n (árvore cheia) e máxima proporcional a n(árvore degenerada).

- A altura indica o esforço computacional necessário para alcançar qualquer nó na árvore (em melhor e pior caso).

Interface do tipo “arvore”

- Portanto, há importância em mantermos as árvores com altura pequena, ou seja, manter a distribuição de nós próxima da árvore cheia.

- Portanto, seria interessante termos uma função que calcule a altura da árvore, pois futuramente será utilizada em árvores AVL.

- Como implementar uma função que calcule a altura de uma árvore?

Interface do tipo “arvore”

- Se árvore for vazia, sua altura é -1.

- Caso não seja vazia, sua altura será de 1 + altura máxima das subárvores esquerda e direita.

```
int arv_altura(Arv *a){
    if(arv_vazia(a)){
        return -1;
    }
    else{
        return 1 + max(arv_altura(a->esq),arv_altura(a->dir));
    }
}
```

Interface do tipo “arvore”

- A função max descobre qual o maior de 2 elementos:

```
int max(int a, int b){
    return ( a > b ) ? a : b;
}
```

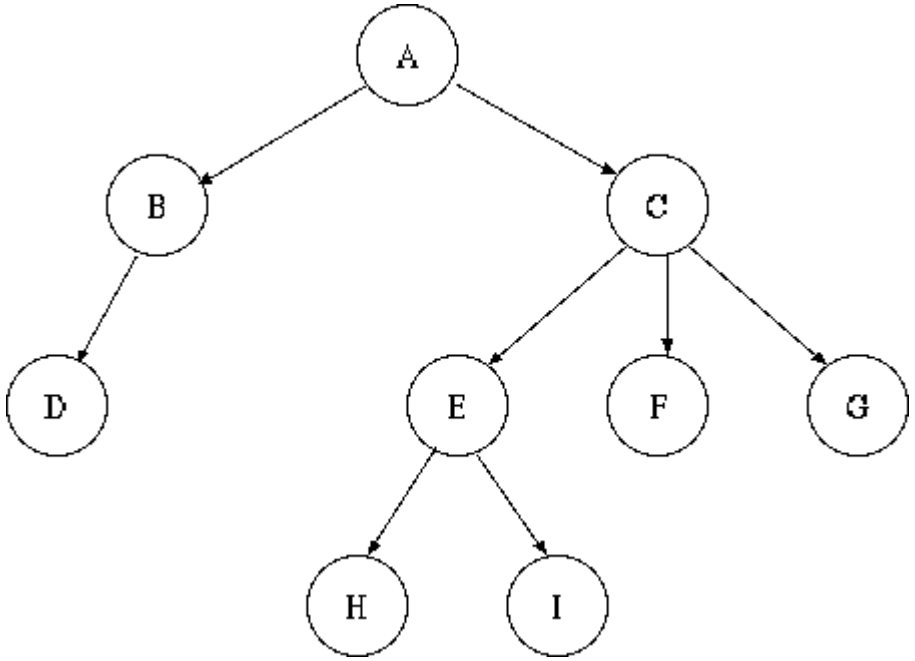
Tópicos

- ~~Conceitos básicos.~~
- ~~Árvores Binárias.~~
- ~~Representação em C.~~
- ~~Interface do tipo “arvore”.~~
- **Árvores com número variável de filhos.**
- **Resumo.**

Árvores com n° variável de filhos

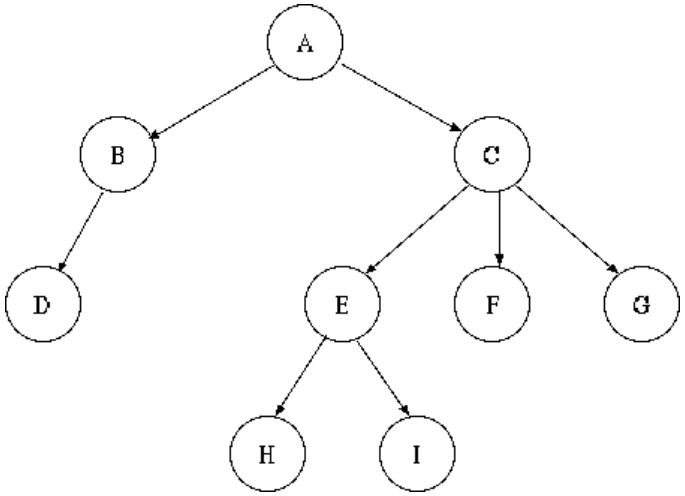
- Agora vamos considerar arvores que tenha um grau maior que 2.

- Agora faz mais sentido falarmos em 1° subarvore (sa_1), 2° subarvore (sa_2), etc.



Árvores com n° variável de filhos

- Podemos representar textualmente usando o seguinte formato genérico:
- <raiz sa_1 sa_2 sa_3... sa_n>



- Como ficaria a representação da árvore acima?
- <a<b<d<><><>>><c<e<h<><><>><i<><><>>><f<><><>><g<><><>>>>

Árvores com n° variável de filhos

- Como ficaria a representação em C de uma árvore com número de filhos diferente de 2?

```
struct arv{
    char info;
    struct arv *sa1, *sa2,  *sa3;
};
```

Cada ponteiro indica um nó de árvores filha.

Árvores com n° variável de filhos

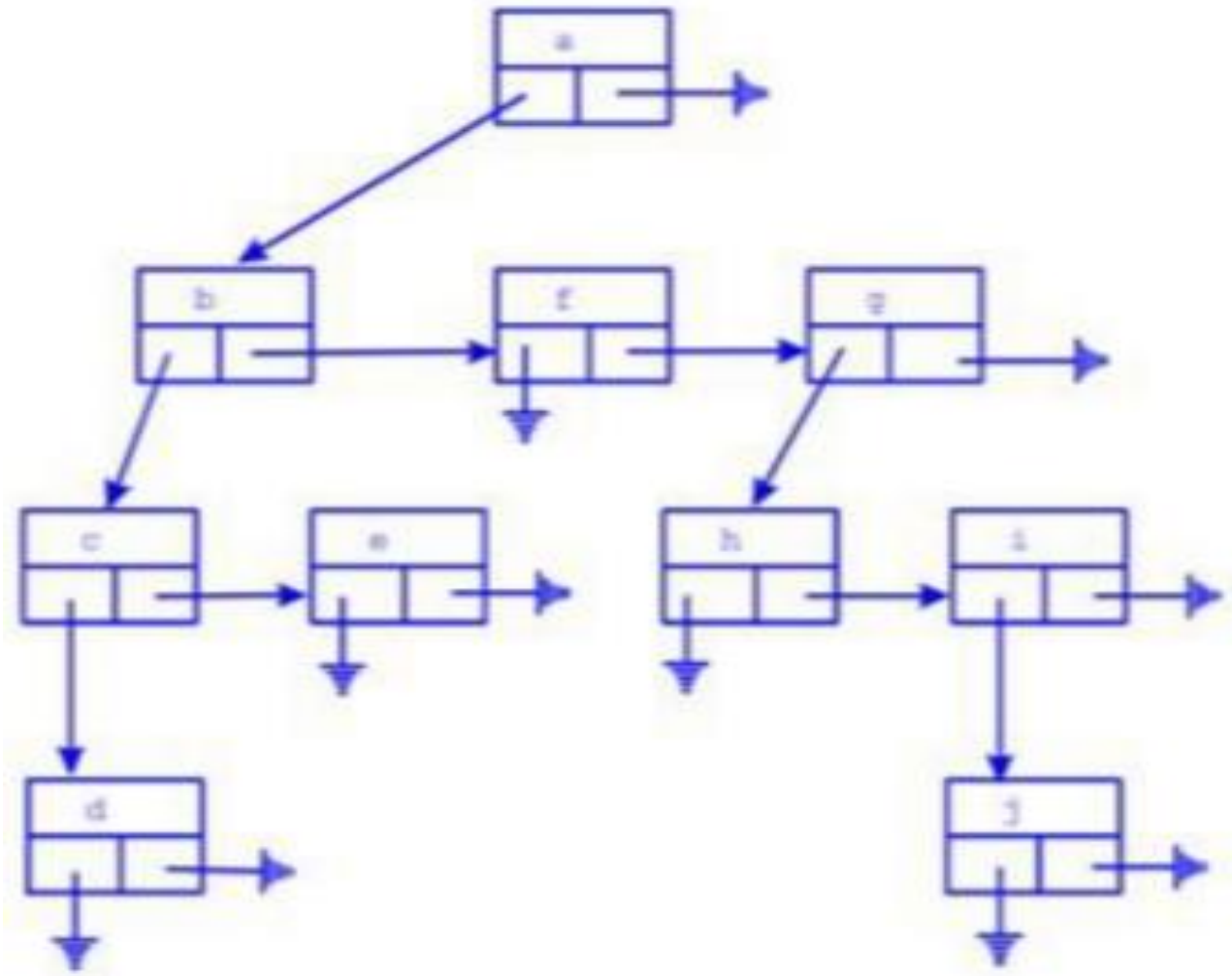
- Para árvores com número variável de filhos, temos:

```
struct arv{
    char info;
    struct arv *prim,
    struct arv *prox;
};
```

Ponteiro para o primeiro filho.

Ponteiro para eventuais irmãos

Árvores com n° variável de filhos



Tópicos

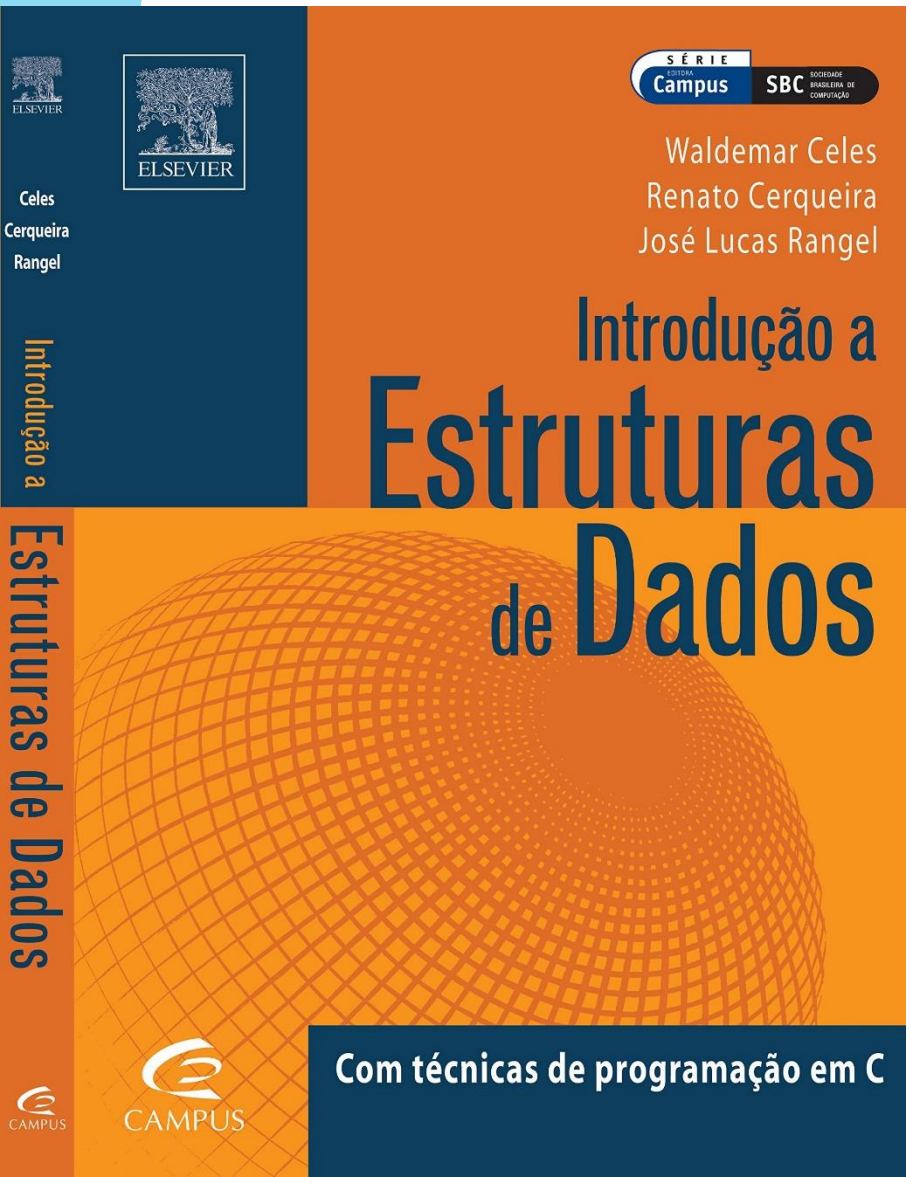
- ~~Conceitos básicos.~~
- ~~Árvores Binárias.~~
- ~~Representação em C.~~
- ~~Interface do tipo “arvore”.~~
- ~~Árvores com número variável de filhos.~~
- **Resumo.**

Resumo

- **Foi demonstrado:**
  - **Conceitos básicos sobre árvores binárias e com n° variável de filhos;**
  - **Características de árvores, ordens de percurso, altura, quantidade de nós, etc.**
  - **Aplicações;**
  - **Implementações em C de árvores;**

Dúvidas ?

Referências



- CELES, Waldemar; CERQUEIRA, Renato; RANGEL, José Lucas. **Introdução a Estruturas de Dados ~~e~~om técnicas de programação em C**. Rio de Janeiro: Elsevier (Campus), 2004. 4ª Reimpressão. 294 p.