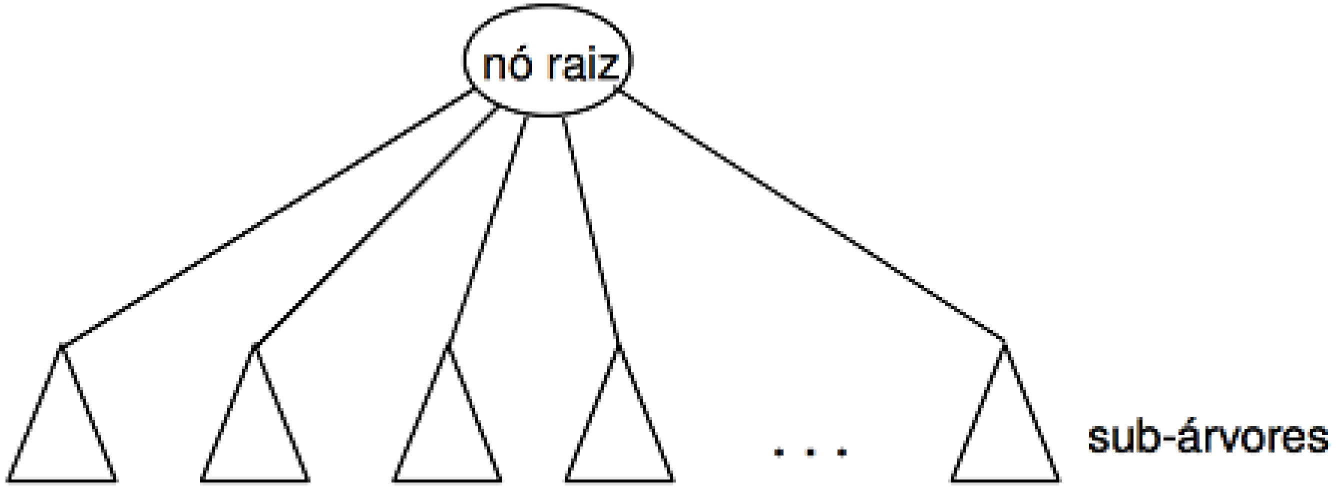
Arvores Binárias

Prof. D.Sc. Saulo Ribeiro

Conceito de Árvore

Um conjunto de nós tal que:

- existe um nó r, denominado **raiz**, com zero ou mais sub-árvores, cujas raízes estão ligadas a r.



Conceito de Árvore

Cronômetro Simples
(Minutos : Segundos)
0 : 0
START STOP
Rotina Acionada 0 Vezes

- os nós raízes destas sub-árvores são os **filhos** de r.
- os **nós internos** da árvore são os nós com filhos.
- as **folhas** ou **nós externos** da árvore são os nós sem filhos.

Exemplos concretos do uso de Árvore

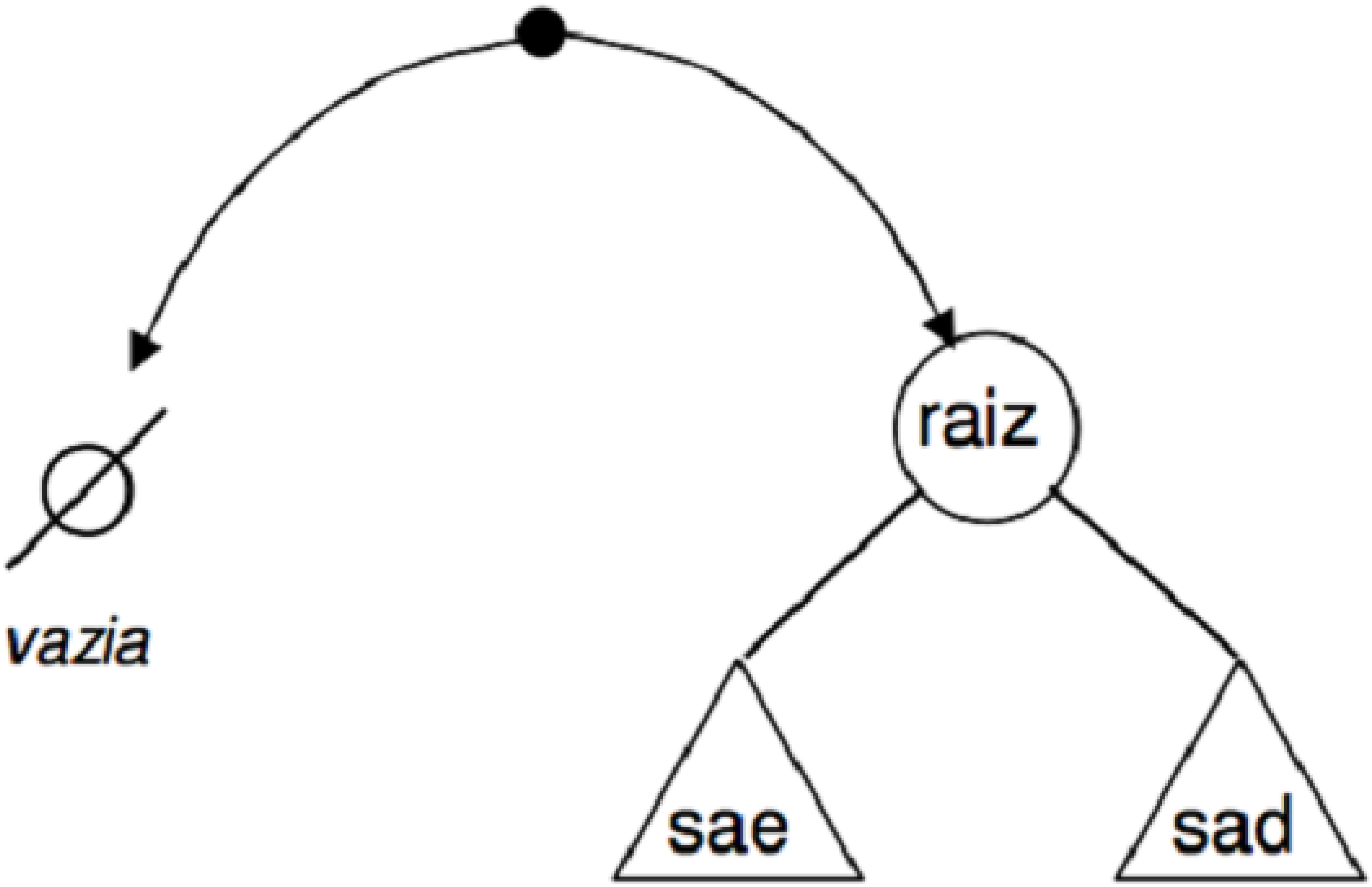
- Estrutura de diretórios e arquivos de um sistema operacional
- Análise semântica de equações matemáticas
- *É uma estrutura usada por vários outros algoritmos...*

Conceito de Árvore Binária

- uma árvore em que cada nodo tem zero, um ou dois filhos.
- uma árvore binária é:
  - ⋆ uma árvore vazia, ou;
  - ⋆ um nodo raiz com duas sub-árvores: a sub-árvore da direita (sad); a sub-árvore da esquerda (sae).

Conceito de Árvore Binária

Cronômetro Simples
(Minutos : Segundos)
0 : 0
START STOP
Rotina Acionada 0 Vezes

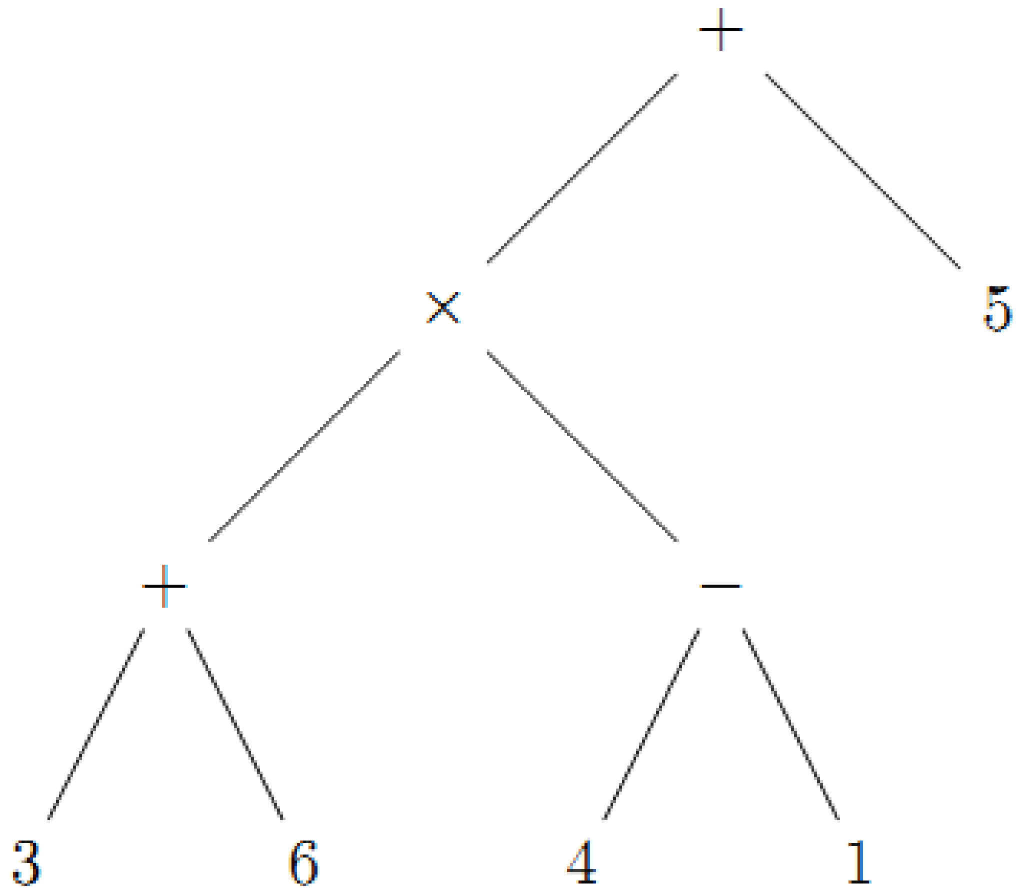


Exemplos de Árvore Binária

Árvore binária representando expressões aritméticas:

- nós folhas representam operandos
- nós internos operadores
- exemplo: $(3+6) \times (4-1) + 5$

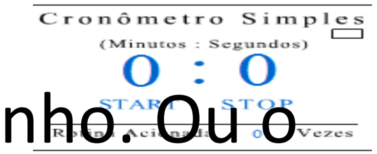
Exemplos de Árvore Binária



Conceitos ligados a Árvore Binária

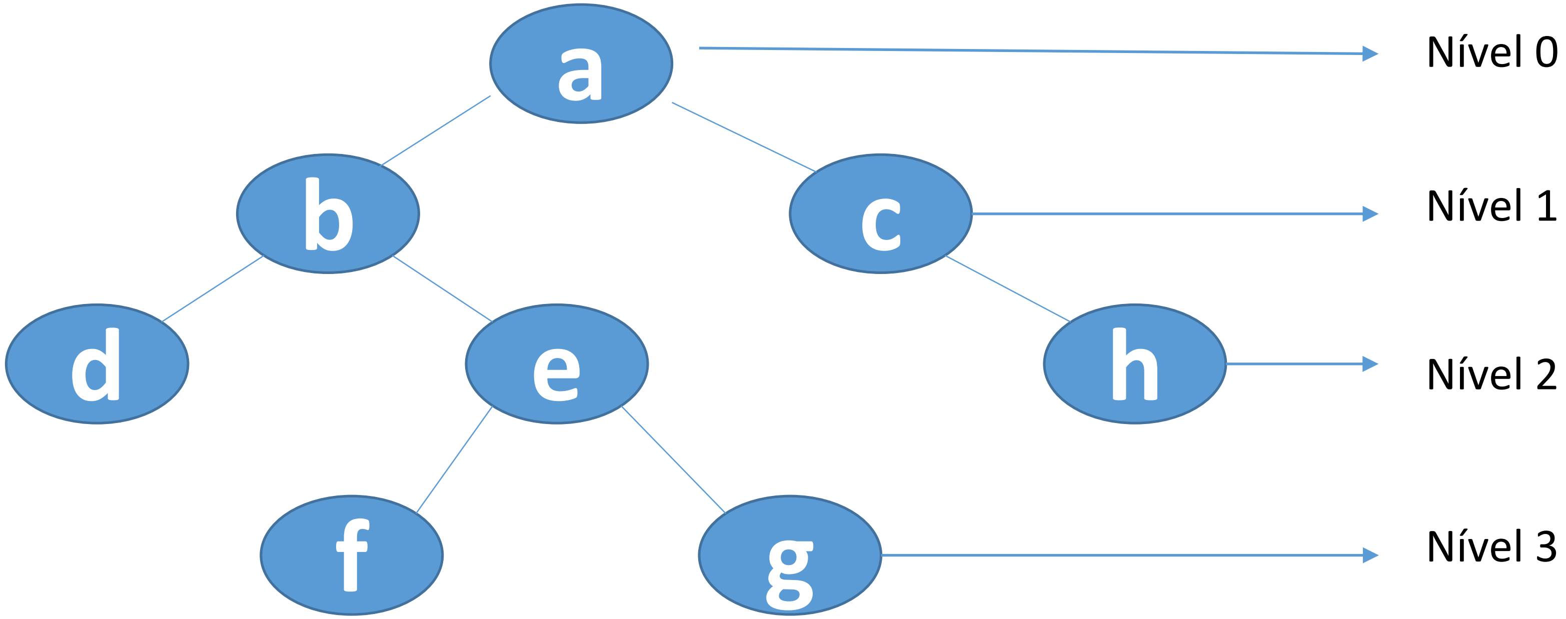
- **Propriedade fundamental de árvores: Nem sempre existe um caminho entre dois elementos de uma árvore, mas se existe, este caminho é único.**
- Um caminho entre dois elementos e1 e e2 é uma sequência <x1, x2,..., xn>, onde e1=x1, é o primeiro elemento, e2=xn é o último elemento, e cada elemento é pai de seu sucessor.
- A longitude de um caminho <x1, x2,..., xn> é N-1, ou seja, o número de vezes que se aplica a relação pai/filho.
- Sempre existe uma caminho de longitude 0 que vai de um elemento r para si mesmo e corresponde a sequência <r>.
- O caminho que parte da raiz e termina numa folha é chamado de galho ou ramo da árvore.

Conceitos ligados a Árvore Binária



- A altura (h) de uma árvore binária é a longitude do maior caminho. Ou o números de vezes que se aplica a relação pai/filho no maior caminho(igual ao número de ramos/galhos).
- A altura de uma folha é 0. Pode se considerar que a altura de uma árvore vazia é -1, para facilitar na implementação.
- O peso de uma árvore é o número de elementos desta árvore. Recursivamente se pode definir como a soma dos pesos das subarvores mais 1.
- De acordo com a definição, o peso de uma árvore vazia é zero.
- O nível de um elemento dentro da árvore binária se define como a longitude do caminho que parte da raiz e chega até esse elemento. Desta forma, o nível da raiz é 0, e o nível de qualquer elemento é o nível de seu pai mais 1.

Conceitos ligados a Árvore Binária



A altura da árvore é 3, que coincide com a altura do nó a.

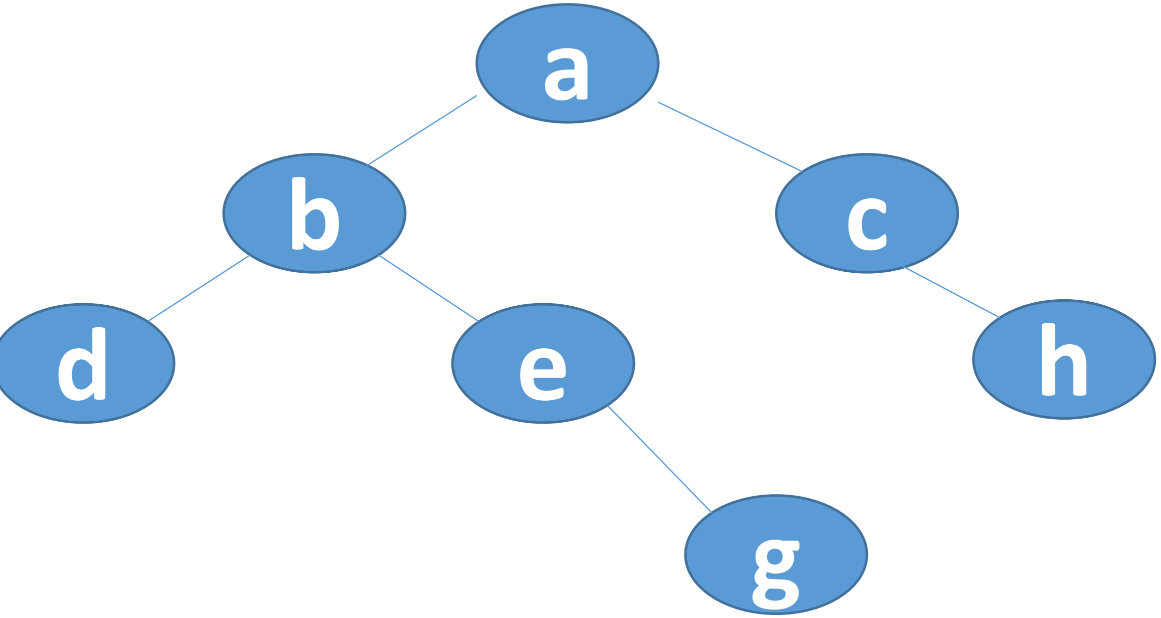
O peso da árvore é 8.

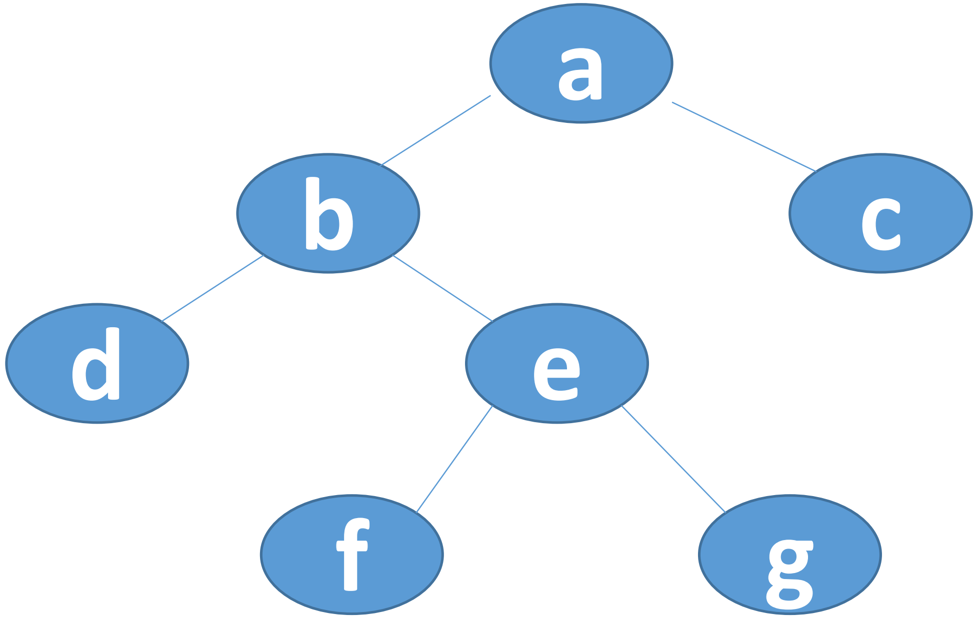
O elemento “a” é a raiz da árvore.

A altura de (d,f,g,h) é 0, de (e,c) é 1, de (b é 2) e de a é 3.

Conceitos ligados a Árvore Binária

- Uma árvore binária é completa, se todo elemento não terminal tem associado exatamente duas subarvores não vazias. Isto é, ou este elemento tem as duas subarvores ou não tem nenhuma.



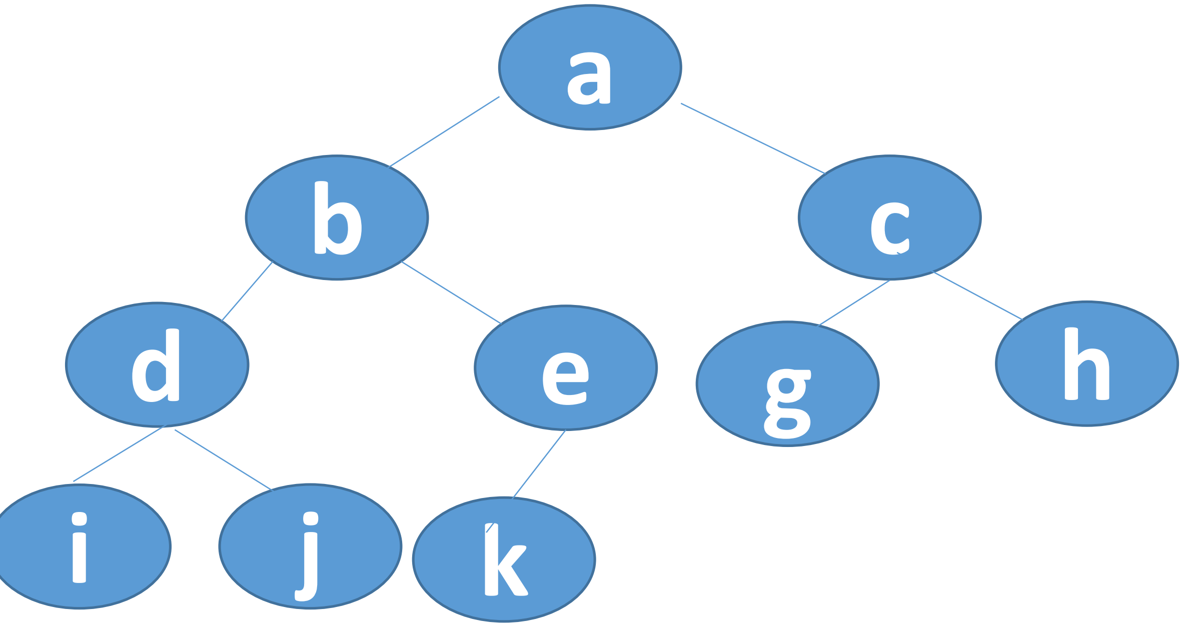


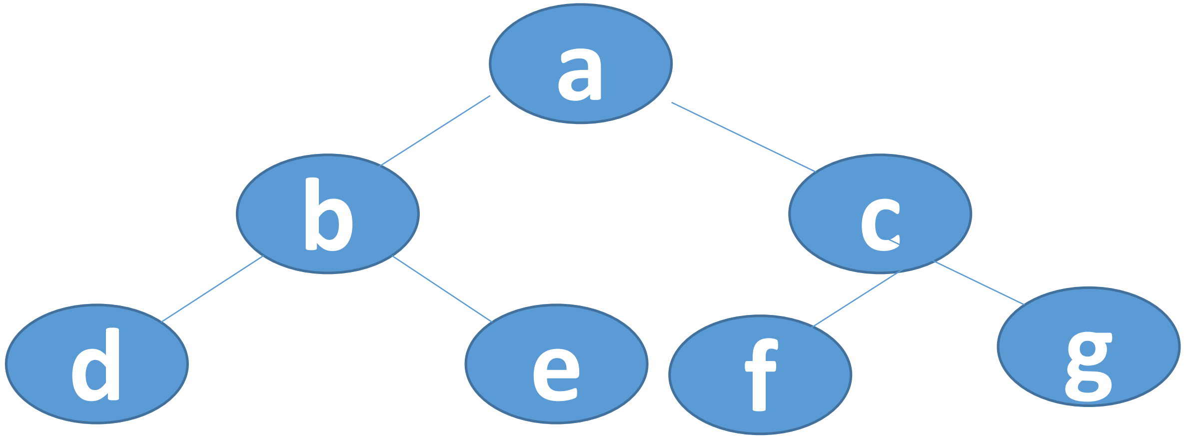
- Arvore binária incompleta

Arvore binária completa

Conceitos ligados a Árvore Binária

- Uma árvore binária é **cheia**, se é **completa** e todas as folhas estão no mesmo nível. Ou seja, todos os nós internos tem duas subarvores associadas. E uma árvore é dita **quase cheia** se está cheia até o penúltimo nível e todas as folhas do seguinte nível, estão o mais a esquerda quanto possível.



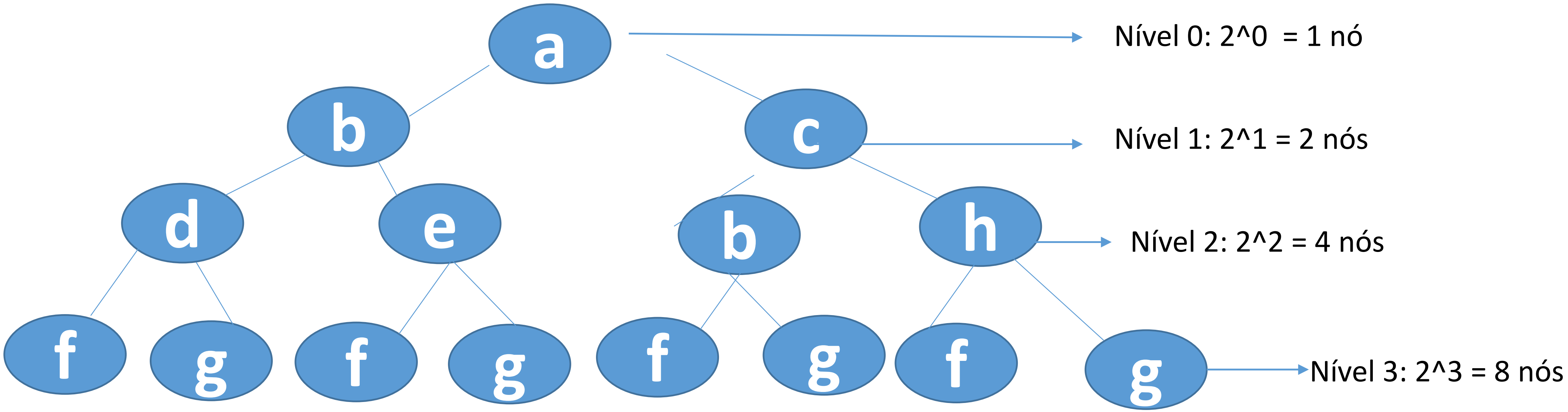


- Arvore binária quase cheia

Arvore binária cheia

Conceitos ligados a Árvore Binária - Altura

Cronômetro Simples
(Minutos : Segundos)
0 : 0
START STOP
Rotina Acionada 0 Vezes



O número de N de nós de uma árvore cheia de altura h é:

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

Conceitos ligados a Árvore Binária - Altura

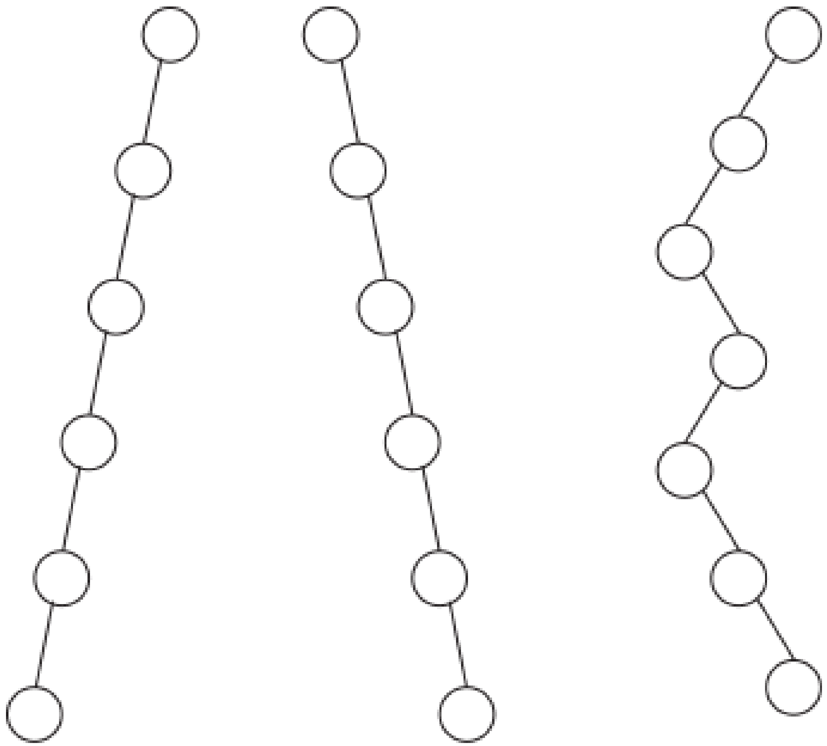


FIGURA 3.8 Árvores zigue-zague.

Se a árvore está degenerada(os nós internos tem somente uma subarvore)
O número de N de nós de uma árvore degenerada de altura h é:

$$N - 1 = h$$

Conceitos ligados a Árvore Binária - Altura

Cronômetro Simples
(Minutos : Segundos)
0 : 0
START STOP

A relação entre a altura de uma árvore binária e o seu número de nós é um dado importante para várias aplicações. Para um valor fixo de *n*, indagar-se-ia quais são as árvores binárias que possuem altura *h* máxima e mínima. A resposta ao primeiro problema é imediata. A árvore binária que possui altura máxima é aquela cujos nós interiores possuem exatamente uma subárvore vazia. Essas árvores são denominadas *zigue-zague* e encontram-se ilustradas na Figura 3.8. Naturalmente, a altura de uma árvore zigue-zague é igual a *n*. Por outro lado, uma árvore completa sempre apresenta altura mínima, conforme é visto a seguir.

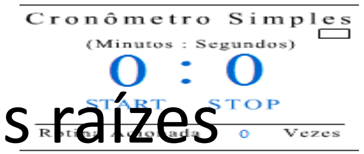
- Lema 1: Seja T uma árvore binária completa com $N>0$ nós. Então T possui altura mínima. Além disso, h = piso de logN

Conceitos ligados a Árvore Binária - Altura

Cronômetro Simples
(Minutos : Segundos)
0 : 0
START STOP
Rotina Acionada 0 Vezes

- Esforço computacional necessário para alcançar qualquer nó da árvore
  - proporcional à altura da árvore
  - altura de uma árvore binária com *n* nós
    - mínima: proporcional a *log n* (caso da árvore cheia)
    - máxima: proporcional a *n* (caso da árvore degenerada)

Conceitos ligados a Árvore Binária



- Duas árvores binarias são **iguais** se ambas são vazias, ou se suas raízes são iguais, e o mesmo acontece para suas respectivas subarvores esquerda e direita.
- Duas árvores são **isomorfas** se têm a mesma forma(estrutura), mas não necessariamente os mesmos elementos.
- Duas árvores são **semelhantes** se têm os mesmos elementos, ainda que não sejam isomorfas.

Funções do TAD Arbin

- Arbin inicArbin(void) : cria e retorna uma arvore binaria vazia
- Arbin esqArbin(Arbin a): retorna a subarvore esquerda
  - Pre: a!=arvore vazia
  - Pos: esqArbin = subarvore esquerda
- Arbin dirArbin(Arbin a): retorna a subarvore direita
  - Pre: a!=arvore vazia
  - Pos: dirArbin = subarvore direita
- TipoA raizArbin(Arbin a): retorna a raiz
  - Pre: a!=arvore vazia
  - Pos: raizArbin = elem

Funções do TAD Arbin

- void destruirArbin(Arbin a): destrói a arvore binária, retornando toda a memória ocupada.

Declaração das estruturas do TAD Arbin

Cronômetro Simples

- typedef int TipoA;

- typedef struct NodoArbin{

  TipoA info;

  struct NodoArbin *esq, *dir;

}Tarbin, *Arbin;

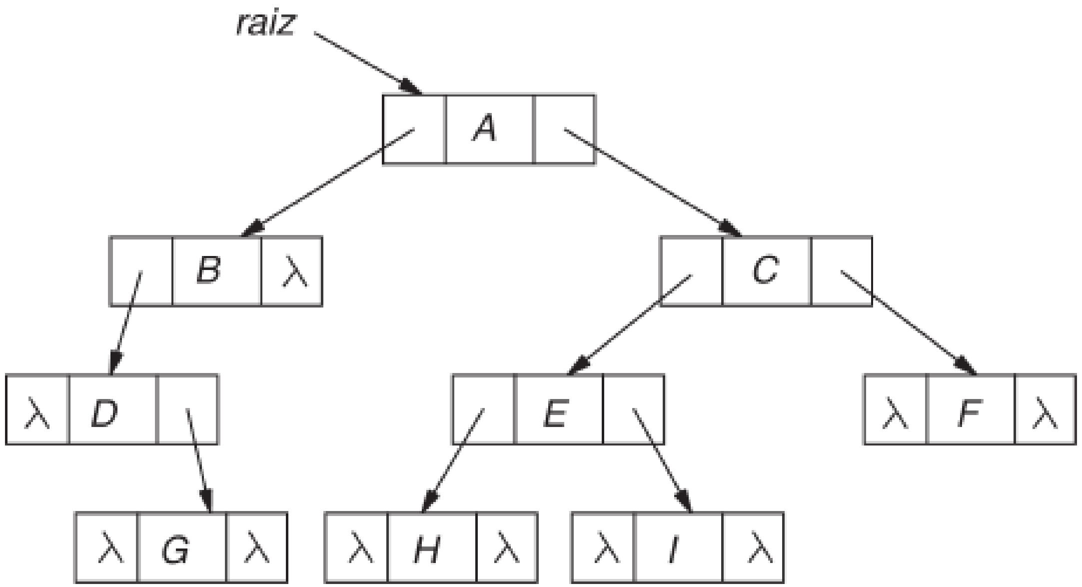


FIGURA 3.11 Armazenamento de uma árvore binária.

A representação de uma árvore binária é feita através de uma referência para o nó raiz da árvore.

Caminhamento em Arvore Binaria

- Pre-ordem: visita a raíz, percorre em pre-ordem a subarvore esquerda e depois percorre em pre-ordem a subarvore direita.
- In-ordem: percorre em in-ordem a subarvore esquerda, visita a raíz e depois percorre em in-ordem a subarvore direita.
- Pos-ordem: percorre em pos-ordem a subarvore esquerda e depois a subarvore direita, e por último visita a raiz.
- Níveis: visita a raíz da árvore, os elementos do nível 1 da esquerda para direita, seguidos dos elementos do nível 2, e assim sucessivamente até visitar todos os elementos.

Caminhamento em Arvore Binaria

- Pre-ordem: visita a raíz, percorre em pre-ordem a subarvore esquerda e depois percorre em pre-ordem a subarvore direita.

Implementação do caminhamento em Pre-ordem.

```
void preOrdem(Arbin a){
        if(!vaziaArbin(a)){
                visitaRaiz(raizArbin(a));
                preOrdem(esqArbin(a));
                preOrdem(dirArbin(a));
        }
}
```

Caminhamento em Arvore Binaria

- In-ordem: percorre em in-ordem a subarvore esquerda, visita a raíz e depois percorre em in-ordem a subarvore direita.

Implementação do caminhamento em In-ordem.

```
void inOrdem(Arbin a){
	if(!vaziaArbin(a)){
		inOrdem(esqArbin(a));
		visitaRaiz(raizArbin(a));
		inOrdem(dirArbin(a));
	}
}
```

Caminhamento em Arvore Binaria

- Pos-ordem: percorre em pos-ordem a subarvore esquerda e depois a subarvore direita, e por último visita a raiz.

Implementação do caminhamento em Pos-ordem.

```
void posOrdem(Arbin a){
	if(!vaziaArbin(a)){
		posOrdem(esqArbin(a));
		posOrdem(dirArbin(a));
		visitaRaiz(raizArbin(a));
	}
}
```

Caminhamento em Arvore Binaria

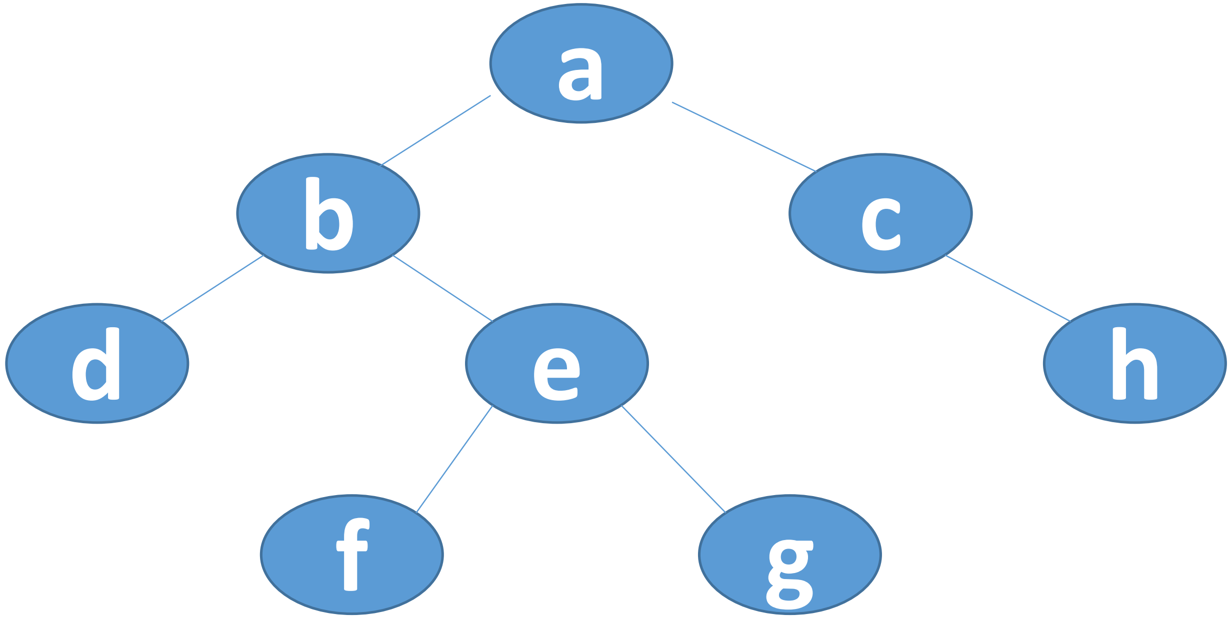
- Níveis: visita a raíz da árvore, os elementos do nível 1 da esquerda para direita, seguidos dos elementos do nível 2, e assim sucessivamente até visitar todos os elementos.

Implementação do caminhamento por Níveis usando uma Fila.

```
void niveisArbin(Arbin a){
        Fila f; Arbin arb;
        if(!vaziaArbin(a)){
                fila = inicFila();
                adicFila(f, a);
                while( !vaziaFila(f)){
                        arb = infoFila(f);
                        elimFila(f);
                        if( !vazioArbin(arb){
                                visitarRaiz(raizArbin(arb));
                                adicFila(f, esqArbin(arb));
                                adicFila(f, dirArbin(abr));
                        }
                }
        }
}
```

Caminhamento em Arvore Binaria

- Pre-ordem: a-b-d-e-f-g-c-h
- In-ordem: d-b-f-e-g-a-c-h
- Pos-ordem:d-f-g-e-b-h-c-a
- Níveis: a-b-c-d-e-h-f-g



Árvore Binária de Busca ou de Pesquisa

Neste capítulo são descritas estruturas de dados adequadas à solução de problemas de busca. Dado um conjunto de elementos, onde cada um é identificado por uma chave, o objetivo é localizar nesse conjunto o elemento correspondente a uma chave específica procurada. Em capítulos anteriores, foram examinados métodos diferentes para resolver esse problema, como busca linear e linear ordenada. No presente capítulo serão vistos métodos de solução que empregam determinados tipos de árvores como estruturas nas quais se processa a busca. Ou seja, os elementos do conjunto são previamente distribuídos pelos nós de uma árvore de forma conveniente. A localização da chave desejada é então obtida através de um caminhamento apropriado na árvore.

Árvore Binária de Busca ou de Pesquisa

É importante ressaltar, mais uma vez, a relevância desse problema na área de computação, em especial nas aplicações não numéricas. Sem dúvida, a operação de busca é uma das mais frequentemente realizadas. Vários métodos de solução empregam árvores como estrutura de armazenamento das chaves. Neste capítulo são examinados dois desses métodos: a árvore binária de busca propriamente dita e a árvore de partilha. O primeiro deles é estudado na Seção 4.2. Após a introdução dos conceitos básicos, essa seção apresenta os algoritmos para busca e inserção em árvores binárias de busca. Em seguida, é considerado o caso real em que as chaves que compõem o problema da busca podem possuir frequências de acesso distintas. Para esse caso, é descrito um algoritmo que constrói a árvore ótima. Finalmente, uma solução mais sofisticada para o problema é descrita na Seção 4.3: árvore de partilha.

Árvore Binária de Busca(ABB) ou de Pesquisa

Cronômetro Simples
(Minutos : Segundos)
0 : 0
START STOP
Rotina Acionada 0 Vezes

Seja $S = \{s_1, \ldots, s_n\}$ o conjunto de chaves satisfazendo $s_1 < \ldots < s_n$. Seja x um valor dado. O objetivo é verificar se $x \in S$ ou não. Em caso positivo, localizar x em S, isto é, determinar o índice j tal que $x = s_j$.

Para resolver esse problema, emprega-se uma árvore binária rotulada T, com as seguintes características:

(i) T possui n nós. Cada nó v corresponde a uma chave distinta $s_j \in S$ e possui como rótulo o valor $rt(v) = s_j$..

(ii) Seja um nó v de T. Seja também v_1, pertencente à subárvore esquerda de v. Então

$$rt(v_1) < rt(v).$$

Analogamente, se v_2 pertence à subárvore direita de v,

$$rt(v_2) > rt(v).$$

A árvore T denomina-se *árvore binária de busca* para S. Naturalmente, se $|S| > 1$, existem várias árvores de busca para S. A Figura 4.1 ilustra duas dessas árvores para o conjunto $\{1, 2, 3, 4, 5, 6, 7\}$.

Resumindo: Uma árvore binária é dita uma árvore binária de busca, se todos os elementos da subarvore esquerda são menores que a raiz e todos os elementos da subarvore direita são maiores que a raiz.

Uma propriedade importante da ABB é que no caminhamento In-ordem se visita sempre os elementos em ordem ascendente.

Árvore Binária de Busca ou de Pesquisa

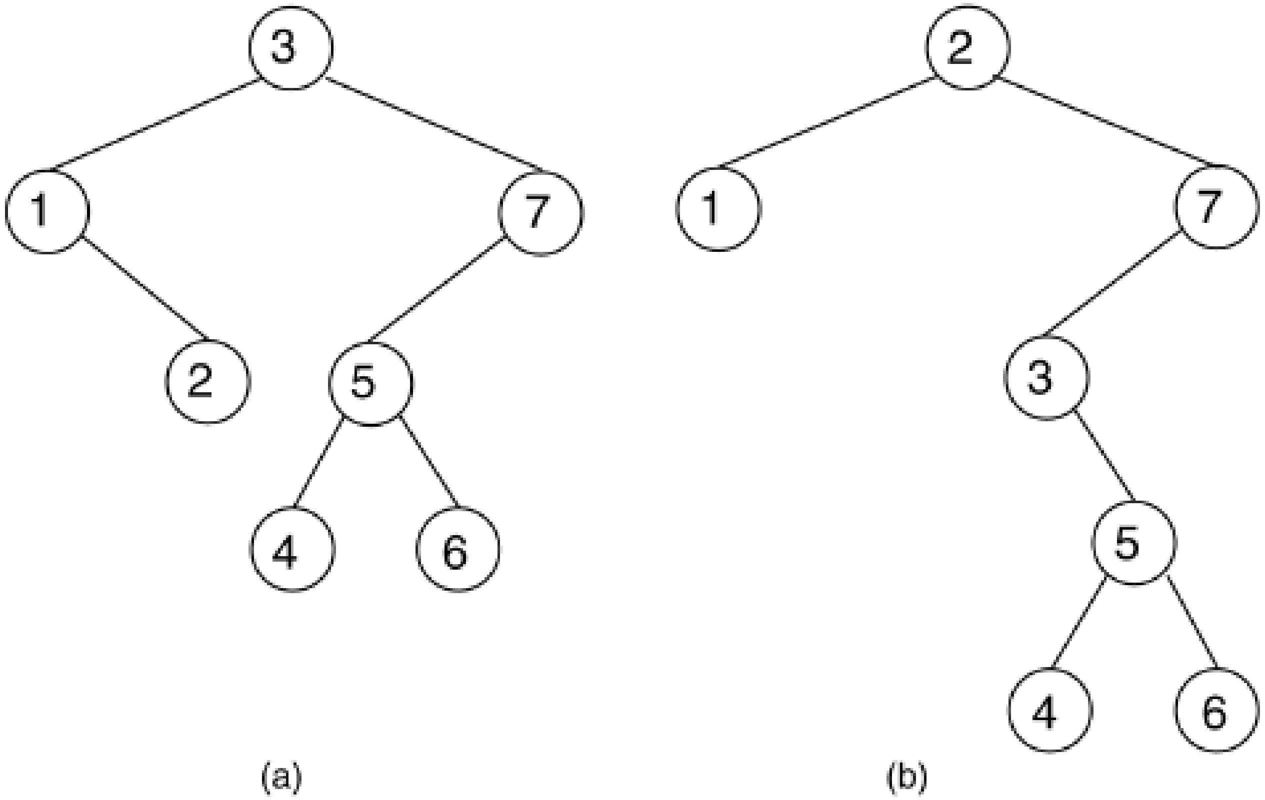
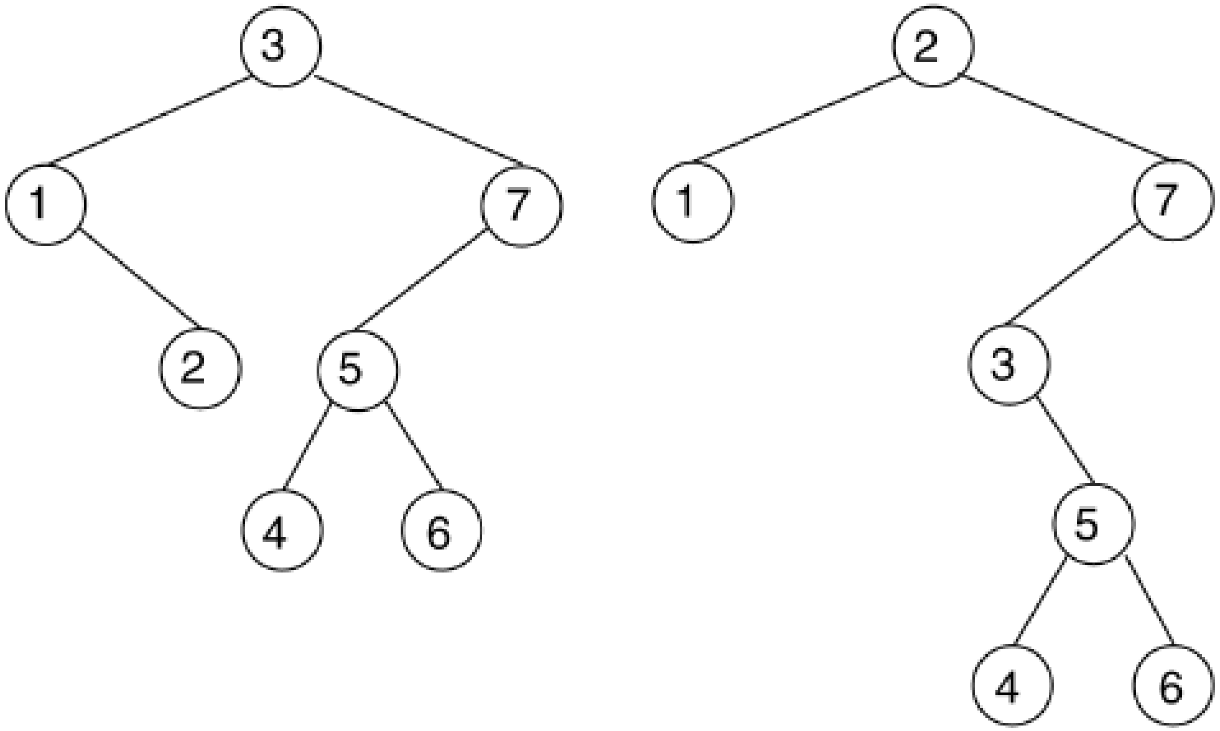


Figura 4.1 Árvores binárias de busca.

Árvore Binária de Busca ou de Pesquisa - Busca

Cronômetro Simples

O algoritmo seguinte implementa a ideia. Suponha que a árvore esteja armazenada da forma habitual, isto é, para cada nó *v*, *esq* e *dir* designam os campos que armazenam ponteiros para os filhos esquerdo e direito de *v*, respectivamente. A raiz da árvore é apontada por *ptraiz*. A variável *f* designa a natureza final da busca. Tem-se então



(a) (b)

Figura 4.1 Árvores binárias de busca.

$f = 0$, se a árvore é vazia.

$f = 1$, se $x \in S$. Nesse caso, *pt* aponta para o nó procurado.

$f > 1$, se $x \notin S$.

Árvore Binária de Busca ou de Pesquisa - Busca

Cronômetro Simples
(Minutos : Segundos)
0 : 0
START STOP
Rotina Acionada 0 Vezes

- **Algoritmo 4.1** **Busca em árvore binária de busca**

```
procedimento busca-arvore(x, pt, f)
    se pt = λ então f := 0
    senão  se x = pt↑.chave então f := 1
           senão  se x < pt↑.chave então
                  se pt↑.esq = λ então f := 2
                  senão pt := pt↑.esq
                        busca-arvore(x, pt, f)
                  senão  se pt↑.dir = λ então f := 3
                         senão pt := pt↑.dir
                               busca-arvore(x, pt, f)
pt := ptraiz;  busca-arvore(x, pt, f)
```

Árvore Binária de Busca ou de Pesquisa - Busca

- **Busca na ABB ou está na ABB:**
- A operação de busca se aproveita da estrutura da ABB, baixando unicamente pelas subarvores nas quais existe a possibilidade de encontrar o elemento buscado.
- No pior dos casos, o algoritmo tem complexidade O(N), onde N é o número de elementos da árvore(seu peso) e sua altura máxima pode ser N (arvore degenerada). Ou o elemento não está na árvore.
- Se a árvore está balanceada(cheia) então no pior caso a busca é proporcional a altura da arvore (log N).

Árvore Binária de Busca ou de Pesquisa - Busca

- **Busca na ABB ou está na ABB:**

```
int estaArbinBusca(Arbin a , TipoA elem){
        if(vaziaArbin(a)) return 0;
        else if (raizArbin(a) == elem) return 1;
        else if ( elem < raizArbin(a) )
                return (estaArbinBusca(esqArbin(a), elem));
        else
                return (estaArbinBusca(dirArbin(a), elem));
}
```

Árvore Binária de Busca ou de Pesquisa - Busca

Cronômetro Simples
(Minutos : Segundos)

Para determinar a complexidade desse algoritmo, basta observar que, em cada passo, isto é, em cada chamada do procedimento *busca-arvore*, é efetuado um número constante de operações. Assim sendo, a complexidade é igual ao número total de chamadas ocorridas no processo. Esse número é também igual ao número de nós existentes no caminho desde a raiz de T até o nó v onde o processo termina. Em um pior caso, v pode se encontrar a uma distância $O(n)$ da raiz de T. Assim sendo, este valor $O(n)$ constitui a complexidade da busca para uma árvore T genérica.

Da observação anterior, conclui-se que a complexidade da busca, para uma árvore T, é igual (no pior caso) à sua altura. Assim sendo, a eficiência do pior caso do algoritmo será tão maior quanto menor for a altura de T. Portanto, é conveniente tentar uma construção da árvore T, de modo a obtê-la com altura mínima. A árvore que possui essa propriedade, para um conjunto de n chaves, é precisamente a completa, conforme demonstrado no capítulo anterior. Nesse caso, a complexidade do algoritmo é igual a $O(\log n)$. O lema seguinte fornece uma relação entre a altura e o número de nós de uma árvore binária completa.

Lema 4.1

Seja T uma árvore binária completa com n nós e altura h. Então

$$2^{h-1} \leq n \leq 2^{h}-1.$$

PROVA O valor $n = 2^h - 1$ ocorre quando a árvore é cheia. Nesse caso, basta observar que o número de nós em um dado nível é exatamente igual ao dobro do anterior. O valor 2^{h-1} corresponde ao caso em que há exatamente apenas um nó no último nível de T. ■

Árvore Binária de Busca ou de Pesquisa - Inserção

Cronômetro Simples
(Minutos : Segundos)
0 : 0
START STOP
Rotina Acionada 0 Vezes

- **Algoritmo 4.2** **Inserção em árvore binária de busca**

$pt := ptraiz;\ \ busca\text{-}arvore(x, pt, f)$

se $f = 1$ **então**

    “inserção inválida”

senão $ocupar(pt1)$

    $pt1 \uparrow.\ chave := x;\ \ pt1 \uparrow.\ info := novo\text{-}valor$

    $pt1 \uparrow.\ esq := \lambda;\ \ pt1 \uparrow.\ dir := \lambda$

    se $f = 0$ **então** $ptraiz := pt1$

    senão **se** $f = 2$ **então**

            $pt \uparrow.\ esq := pt1$

        senão $pt \uparrow.\ dir := pt1$

Árvore Binária de Busca ou de Pesquisa - Inserção

- **Inserção na ABB:**
- O processo de inserir um elemento na ABB tem a mesma estrutura do processo de busca.
- A ideia é seguir a mesma forma de se baixar pela árvore, e no momento de chegar no ponto onde se reconhece que o elemento não está presente(não se pode descer mais), adicionar exatamente ai o elemento. A inserção será feita numa folha.
- Sua complexidade é a mesma da busca. Se a árvore está balanceada(cheia) é log(N) e se a árvore está degenerada é O(N).

Árvore Binária de Busca ou de Pesquisa - Inserção

- **Inserção na ABB:**

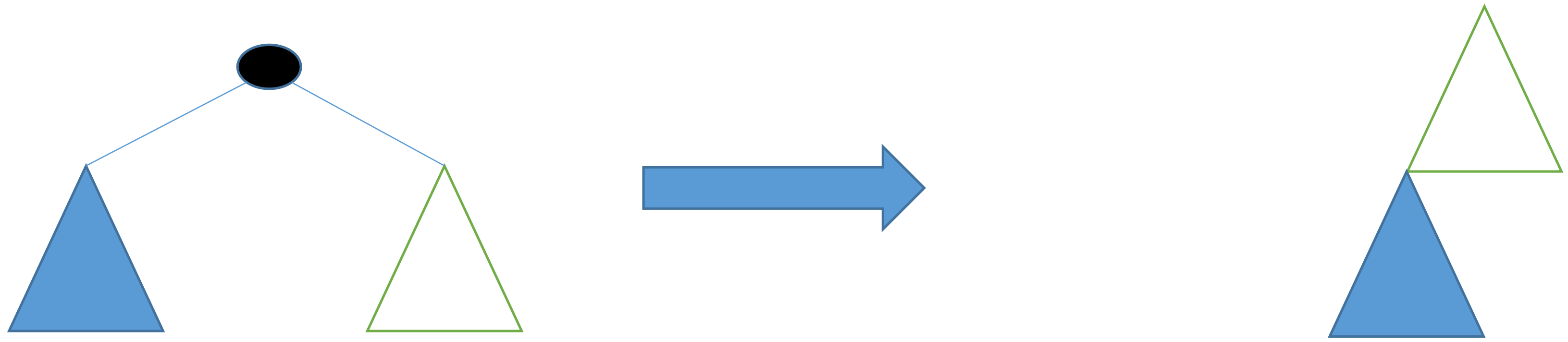
```
Arbin insArbinBusca(Arbin a , TipoA elem){
        if(vaziaArbin(a)){
                a = (Arbin) malloc(sizeof(Tarbin));
                a->info = elem;
                a->esq = a->dir = NULL;
        }
        else if ( elem < raizArbin(a) )
                a->esq = insArbinBusca(a->esq, elem);
        else if(elem != a->info)
                a->dir = insArbinBusca (a->dir, elem);

        return a;
}
```

Árvore Binária de Busca ou de Pesquisa - Eliminação

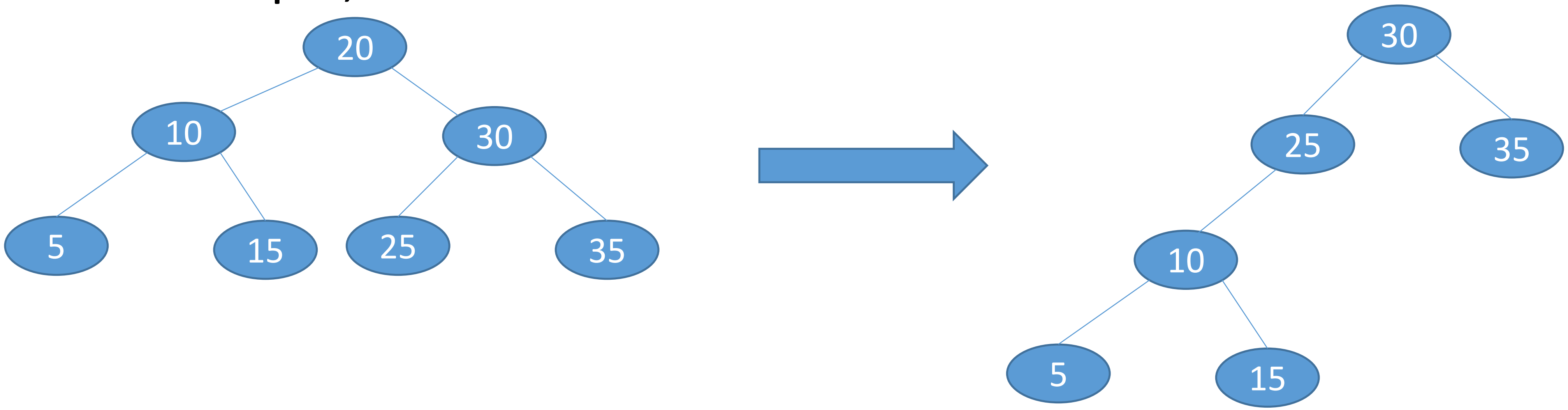
- **Eliminação na ABB:**
- O processo para eliminar um elemento na ABB é mais complicado que os anteriores, porque ao remover um elemento se deve alterar a estrutura da árvore.
- Existem varias formas de fazê-lo, as quais veremos a seguir:

Árvore Binária de Busca ou de Pesquisa - Eliminação

- **Eliminação na ABB:**
- **Caso 1:** Para eliminar o elemento da raiz, se pode colocar a subarvore esquerda a esquerda do menor elemento da subarvore direita.



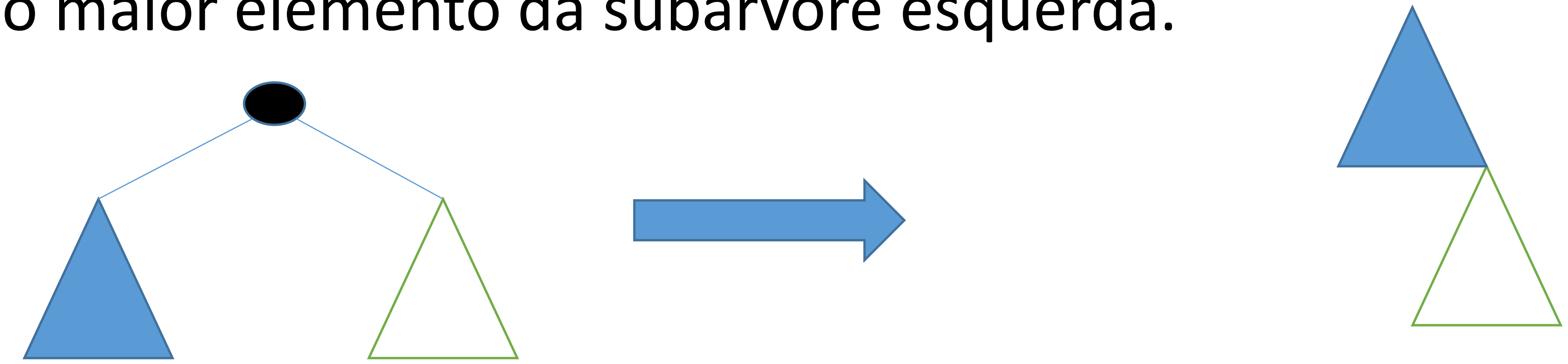
- Por exemplo, ao se eliminar o elemento 20 se tem:



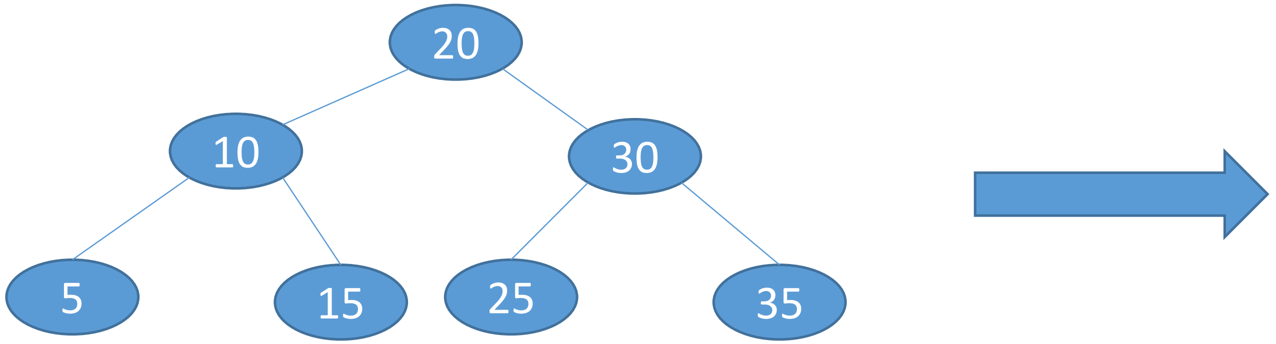
Árvore Binária de Busca ou de Pesquisa - Eliminação

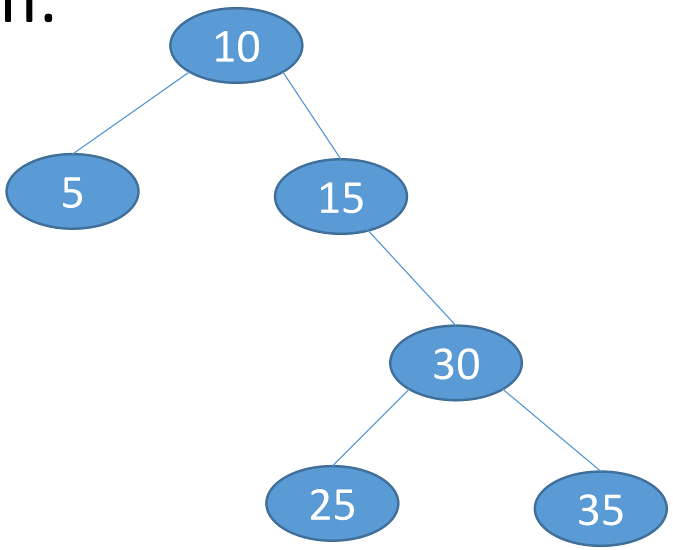
Cronômetro Simples
(Minutos : Segundos)
0 : 0
START STOP
Rotina Acionada 0 Vezes

- **Eliminação na ABB:**
- **Caso 2:** Para eliminar o elemento da raiz, se pode colocar a subarvore direita a direita do maior elemento da subarvore esquerda.



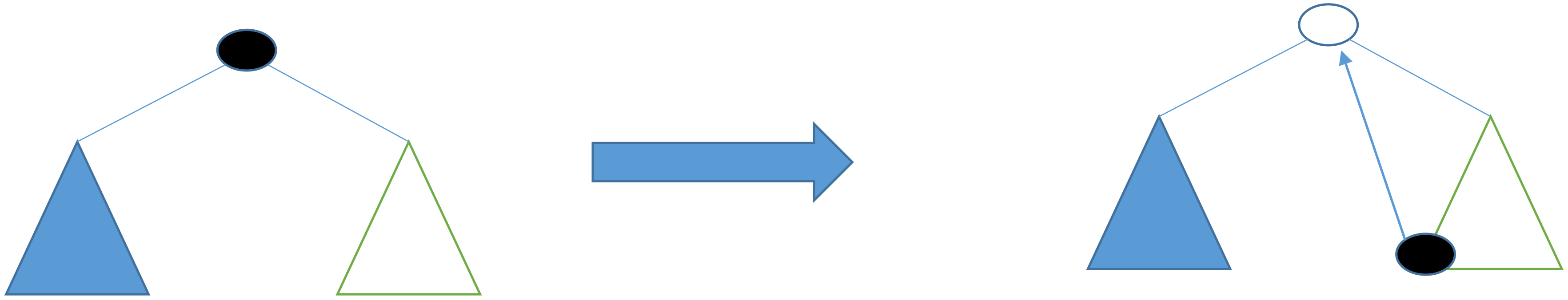
- Por exemplo, ao se eliminar o elemento 20 se tem:



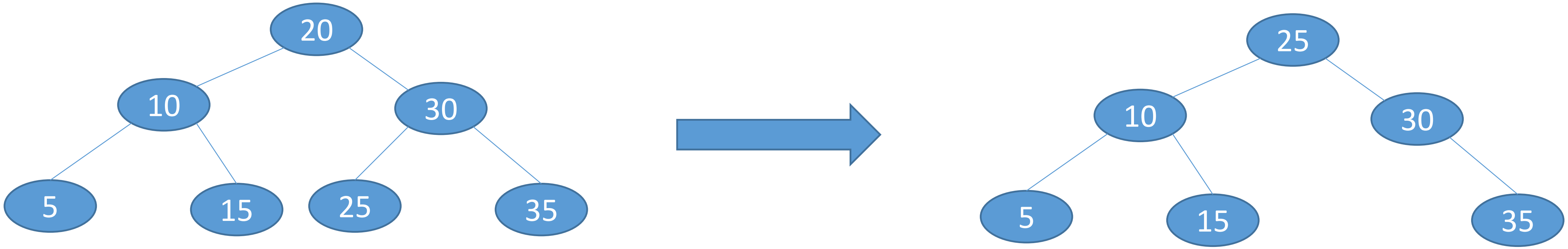


Árvore Binária de Busca ou de Pesquisa - Eliminação

- **Eliminação na ABB:**
- **Caso 3:** Para eliminar o elemento da raiz, se pode substituir a raiz pelo menor elemento da subarvore direita.

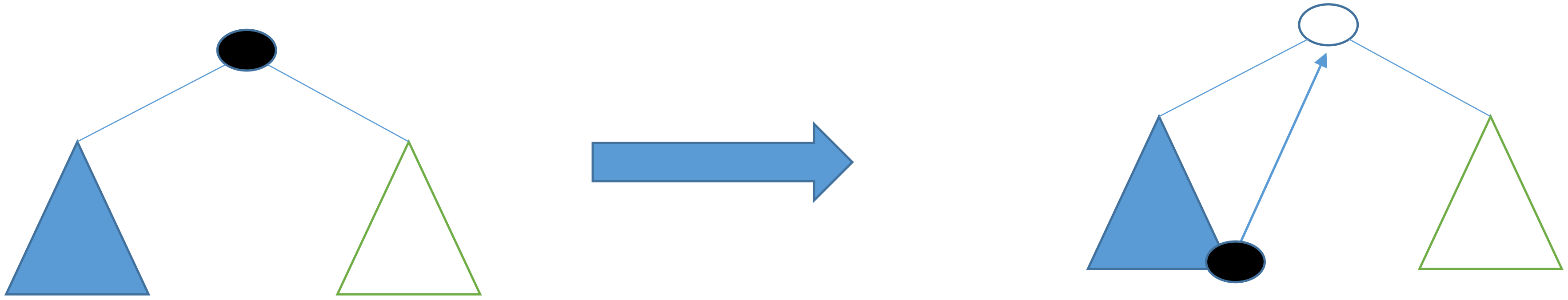


- Por exemplo, ao se eliminar o elemento 20 se tem:

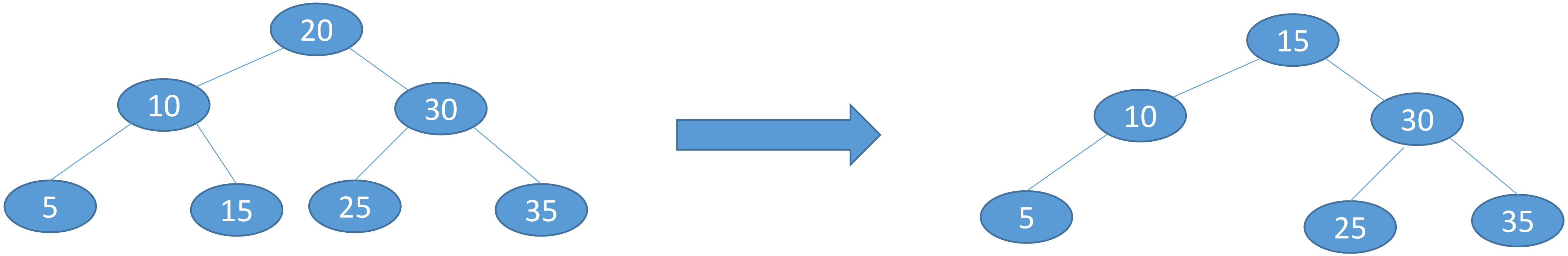


Árvore Binária de Busca ou de Pesquisa - Eliminação

- **Eliminação na ABB:**
- **Caso 4:** Para eliminar o elemento da raiz, se pode substituir a raiz pelo maior elemento da subarvore esquerda.



- Por exemplo, ao se eliminar o elemento 20 se tem:



Árvore Binária de Busca ou de Pesquisa - Eliminação

- **Eliminação na ABB:**
- O algoritmo para implementar a remoção de um elemento utilizando o caso 4, considera três grandes casos:
  - O elemento é a raiz,
  - o elemento está na subarvore esquerda
  - ou o elemento está na subarvore direita.
- No primeira caso(elemento == raiz) se aplica a solução vista anteriormente(caso 4), utilizando uma função que retorna o maior elemento de uma arvore binária.
- Os outros dois casos, fazem a recursão avançar pela subarvore respectiva.

Árvore Binária de Busca ou de Pesquisa - Eliminação

- **Eliminação na ABB:**
- Quando o elemento que se quer eliminar é a raiz, considera-se 3 casos:
- É uma folha: esta é eliminada sem problemas.
- A árvore não tem subarvore esquerda: coloca a subarvore direita no lugar da árvore toda.
- Ou tem ambas as subarvores: busca o maior elemento da subarvore esquerda, e o coloca na raiz, e faz uma chamada recursiva para removê-lo desta subarvore.

```
Arbin elimArbinBusca(Arbin a , TipoA elem){
        Arbin p;   TipoA maior;
        if(raizArbin(a) == elem){
                if(vaziaArbin(esqArbin(a)) && vaziaArbin(dirArbin(a)){
                        free(a);
                        return NULL;
                }
                else if( esqArbin(a) == NULL){
                        p = dirArbin(a);
                        free(a);
                        return p;
                }
                else{
                        maior = maiorElemento(esqArbin(a));
                        a->info = maior;
                        a->esq = elimArbinBusca(esqArbin(a), maior);
                }
        }
        else if( elem < raizArbin(a) )  a->esq = elimArbinBusca(esqArbin(a), elem);
        else
          a->dir = elimArbinBusca(dirArbin(a), elem);
        return a;
}
```

Reconstrução de uma Arbin a partir de seus caminhamentos

- **Suponha que você queira reconstruir a árvore binária sem elementos repetidos, cujos caminhos em ordem e pré-ordem são dados pelas seqüências:**

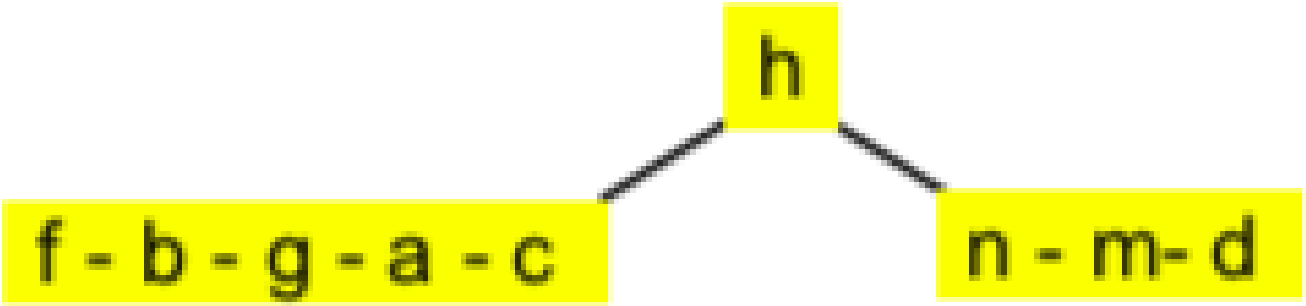
- Pre ordem: h-a-b-f-g-c-m-n-d
- In ordem: f-b-g-a-c-h-n-m-d

Reconstrução de uma Arbin a partir de seus caminhamentos

- **Passo 1: Encontre a raiz e subdivida os caminhamentos. A raiz é sempre o primeiro elemento em pre ordem. Ao localizar o dito elemento em in ordem, os caminhos para as duas subárvores que devem ser associadas à esquerda e à direita são obtidos.**

- Pre ordem:

- In ordem:

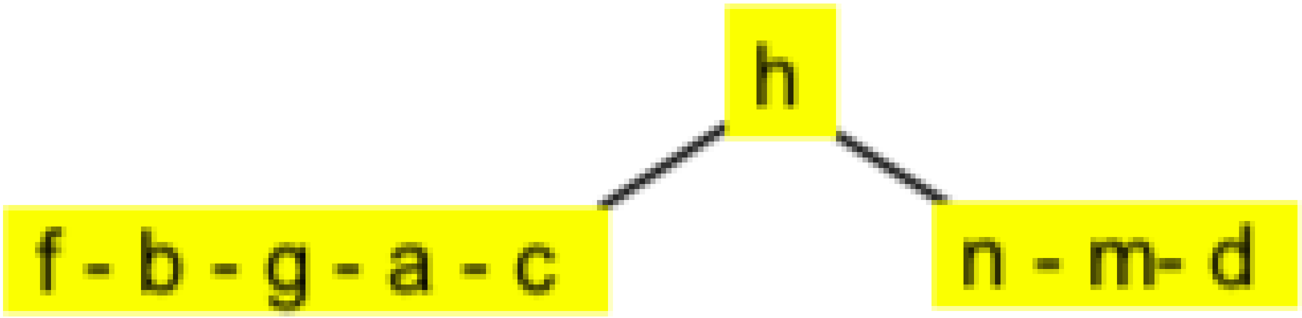


Reconstrução de uma Arbin a partir de seus caminhamentos

- **Passo 2: Conhecendo o peso de cada uma das sub-árvores (5 a esquerda e 3 a direita), é possível calcular o caminhamento em pre ordem de cada uma delas.**

- Pre ordem: h a - b - f - g - c m - n - d

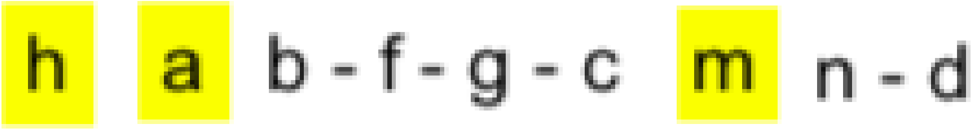
- In ordem:



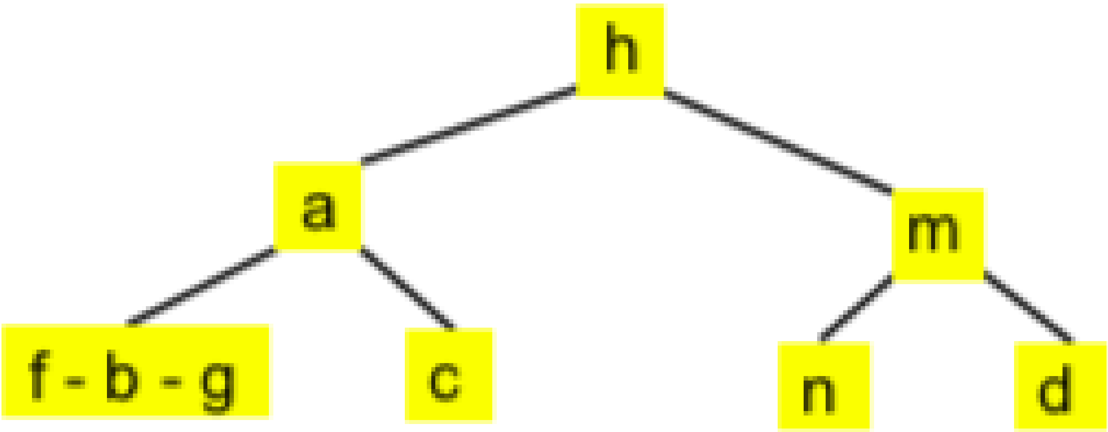
Reconstrução de uma Arbin a partir de seus caminhamentos

- **Passo 3: Repetir o passo 1 em cada sub árvore encontrada.**

- Pre ordem:

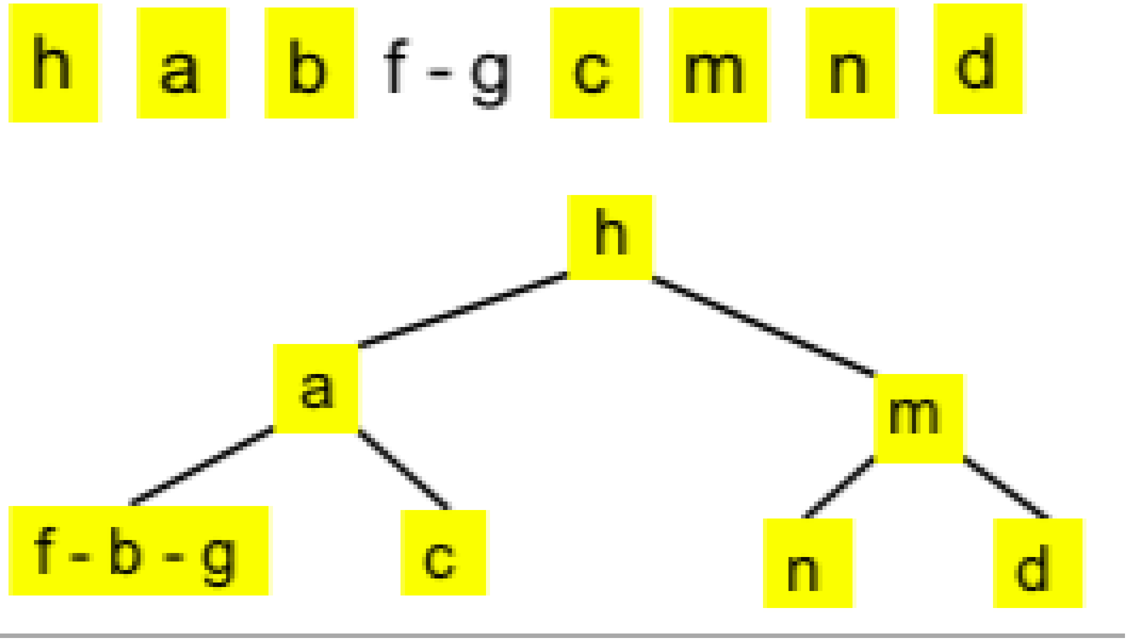


- In ordem:



Reconstrução de uma Arbin a partir de seus caminhamentos

- **Passo 4: Repetir o passo 2 em cada sub árvore encontrada.**

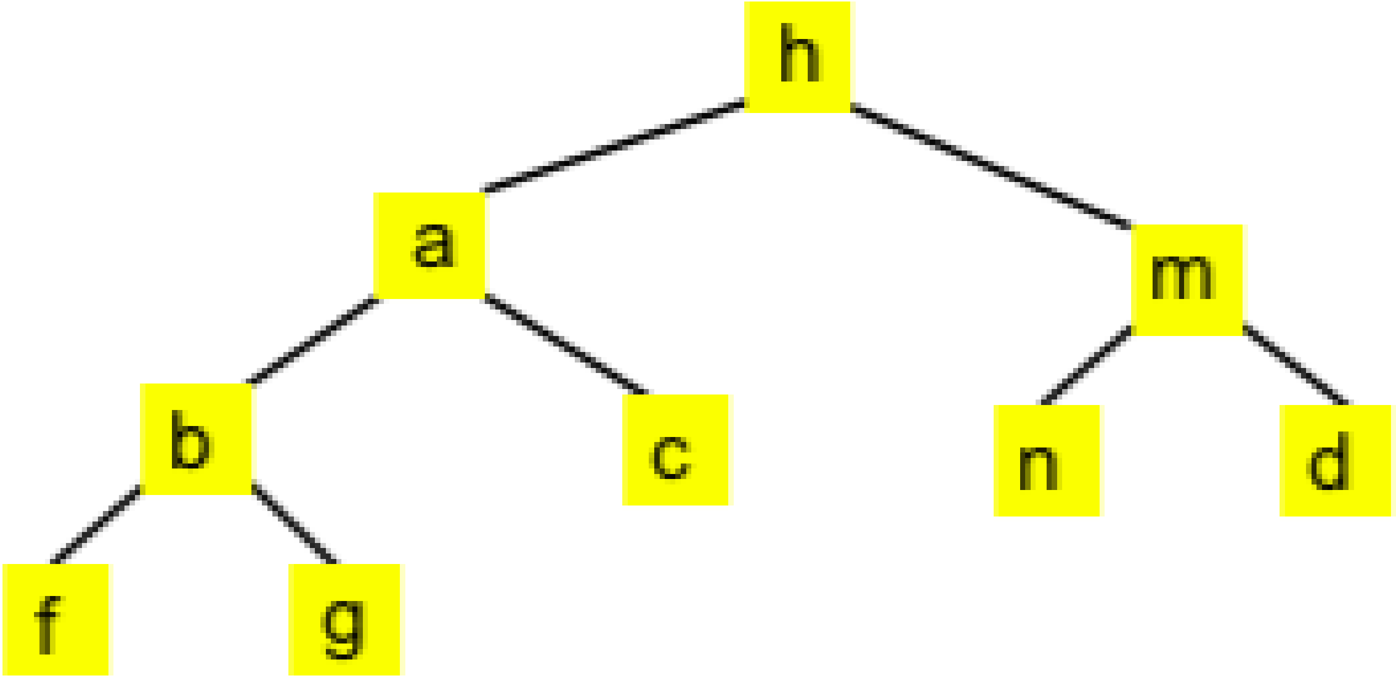


- Pre ordem:

- In ordem:

Reconstrução de uma Arbin a partir de seus caminhamentos

- **Passo 5: Repetir o mesmo processo descrito nos passos 1 e 2 com a única sub árvore que falta.**



- In ordem: