



## Unidade 6 – Balanceamento de Árvores



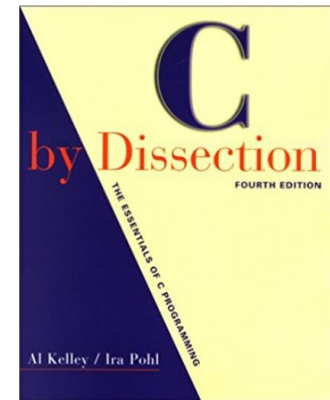
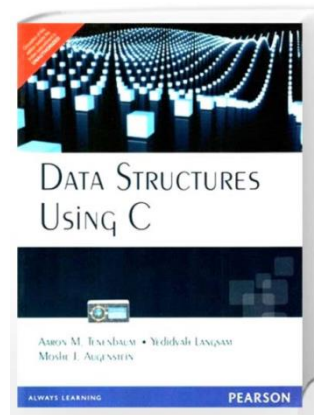
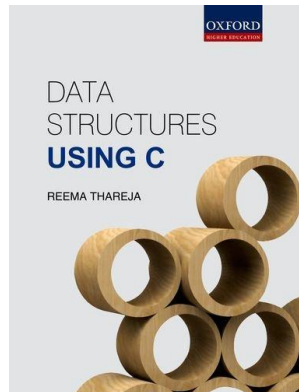
Prof. Aparecido V. de Freitas  
Doutor em Engenharia  
da Computação pela EPUVSP  
[aparecidovfreitas@gmail.com](mailto:aparecidovfreitas@gmail.com)

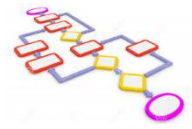




# Bibliografia

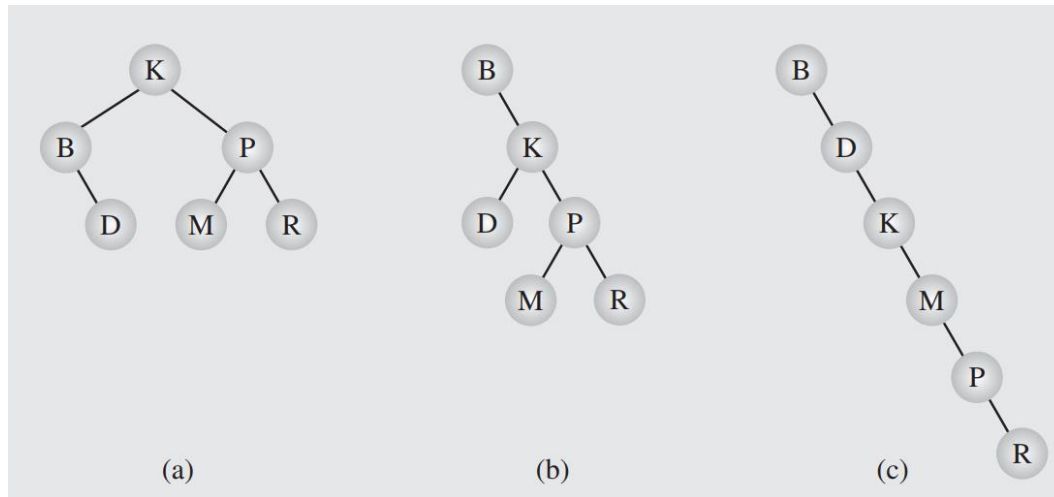
- Data Structures using C – Oxford University Press – 2014
- Data Structures Using C – A. Tenenbaum, M. Augensem, Y. Langsam, Pearson 1995
- C By Dissection – Kelley, Pohn – Third Edition – Addison Wesley





# Balaneando Árvores

- Árvores são estruturas muito apropriadas para representar objetos de forma hierárquica;
- Outra característica importante das árvores é que podem tornar o processo de busca muito rápido;
- No entanto, esta característica associada ao desempenho da busca depende do formato da árvore;
- Por exemplo, as árvores binárias de busca abaixo representam o mesmo conjunto de dados, porém qual delas representa o melhor desempenho em uma operação de busca?



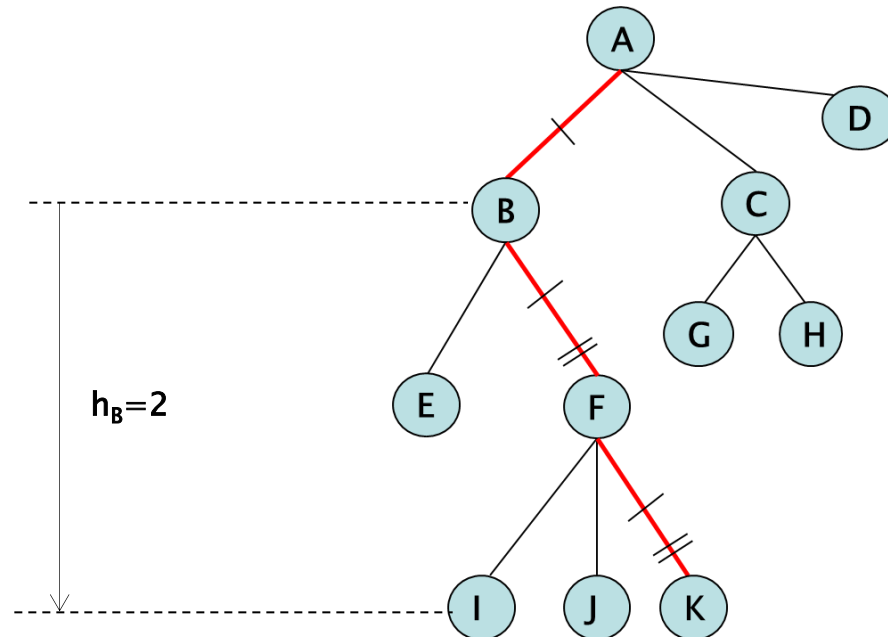
- A árvore da alternativa a) está balanceada e isso favorece o desempenho da busca.





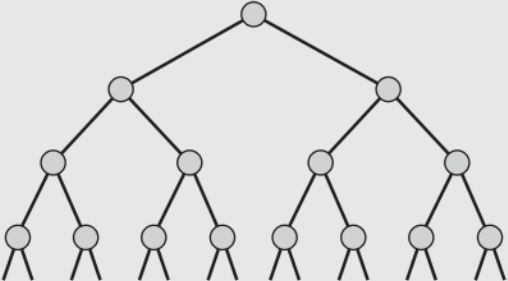
# Árvore Balanceada

- Uma árvore binária em altura, ou simplesmente balanceada, quando a diferença na altura de ambas sub-árvores de qualquer nó é zero ou um;
- Lembrando que a altura de um nó **n** corresponde ao tamanho do caminho do nó n até o seu mais distante descendente. Por exemplo: A altura do nó B é 2.



# Árvore Perfeitamente Balanceada

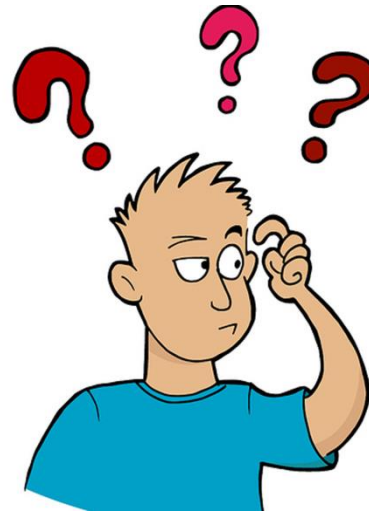
- Uma árvore binária é perfeitamente balanceada se ela é balanceada e todas as suas folhas estão no mesmo nível;

	Nodes at One Level	Nodes at All Levels
	$2^0 = 1$	$1 = 2^1 - 1$
	$2^1 = 2$	$3 = 2^2 - 1$
	$2^2 = 4$	$7 = 2^3 - 1$
	$2^3 = 8$	$15 = 2^4 - 1$
	$2^{10} = 1,024$	$2,047 = 2^{11} - 1$
	$2^{13} = 8,192$	$16,383 = 2^{14} - 1$
	$2^h - 1$	$n = 2^h - 1$





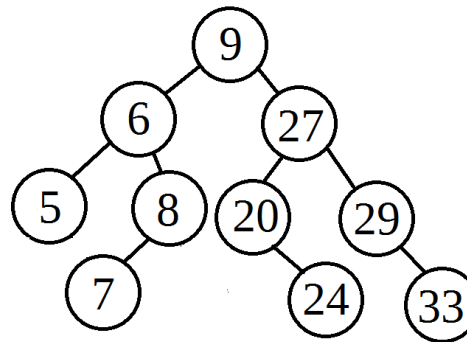
Qual a vantagem de se criar árvores balanceadas?





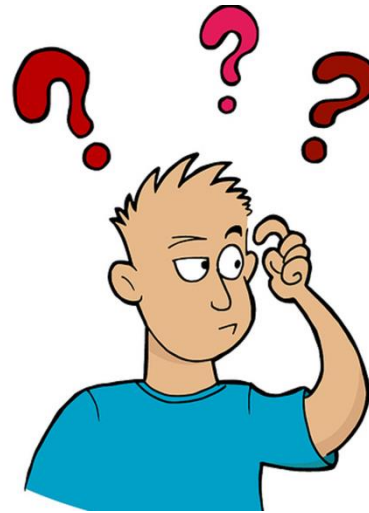
# Árvores Balanceadas

- Considere que uma árvore balanceada tem aproximadamente **10.000** nós;
- Então a árvore tem altura  $\log(10000) \approx 14$ ;
- Em termos práticos, isso significa que, se **10.000** elementos são armazenados em uma árvore balanceada, no máximo **14** nós terão de ser verificados para se localizar um elemento em particular;
- Essa é uma diferença substancial, comparada aos **10.000** testes necessários em uma lista ligada (no pior caso);
- Assim, é muito válido o esforço de se construir árvores balanceadas ou modificar-se uma árvore existente de modo que ela seja balanceada.





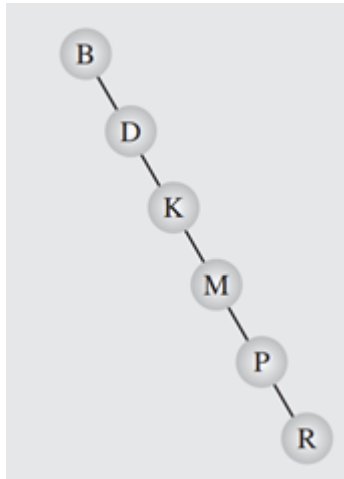
## Como balancear uma árvore binária





# Balanceamento de Árvores Binárias

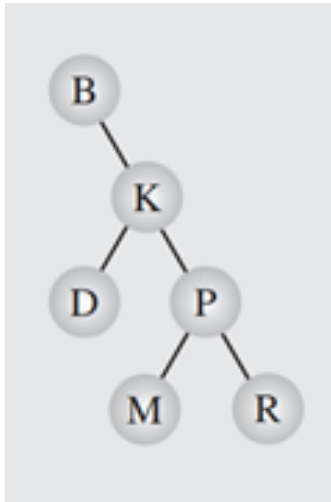
- Há técnicas para se balancear apropriadamente uma árvore binária;
- Algumas baseiam-se em se reestruturar constantemente a árvore quando os elementos chegam;
- Outras consistem em reordenar os próprios dados e então construir a árvore;



- Esta árvore é o resultado de uma corrente de dados particular;
- Quando os dados chegam em ordem ascendente ou descendente, a árvore se parece com uma lista ligada.



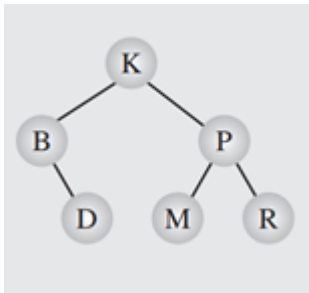
# Balanceamento de Árvores Binárias



- Esta árvore é pendente lateralmente porque o primeiro elemento que chega é a letra B, que precede todas as outras, exceto A;
- Com isso, a sub-árvore da esquerda terá apenas um elemento.



# Balanceamento de Árvores Binárias



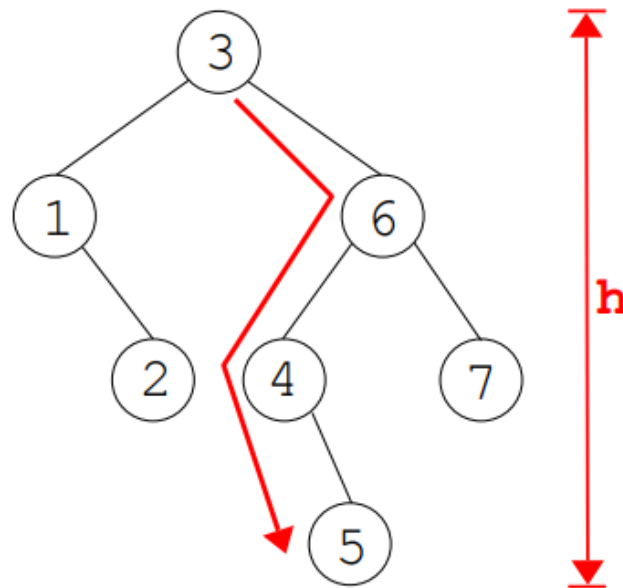
- Esta árvore se parece bem do ponto de vista de balanceamento;
- A raiz contém um elemento próximo do meio de todos os possíveis elementos e P está mais ou menos no meio de K e de Z;
- Isso nos leva a um algoritmo baseado na técnica binária de busca.





# Complexidade de Busca em Árvore Binária

- Busca em árvore binária = caminho da raiz até chave desejada (ou até uma folha, caso chave não exista).



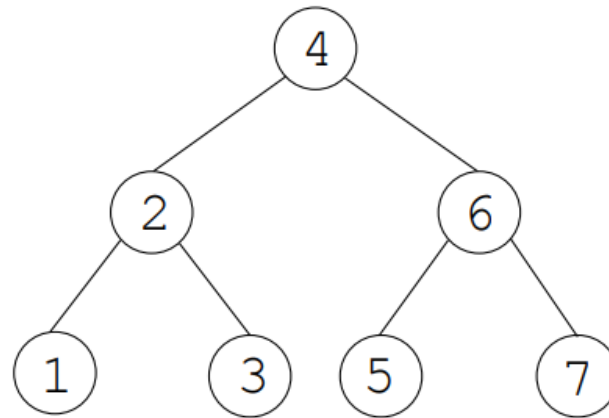
- Pior caso: maior caminho da raiz até folha = altura da árvore





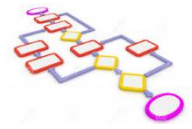
# Complexidade de Busca em Árvore Binária Ótima

- ▶ Árvore ótima: minimiza tempo de busca (no pior caso)
- ▶ Árvore completa, altura:  $h = \lfloor \log n \rfloor + 1$



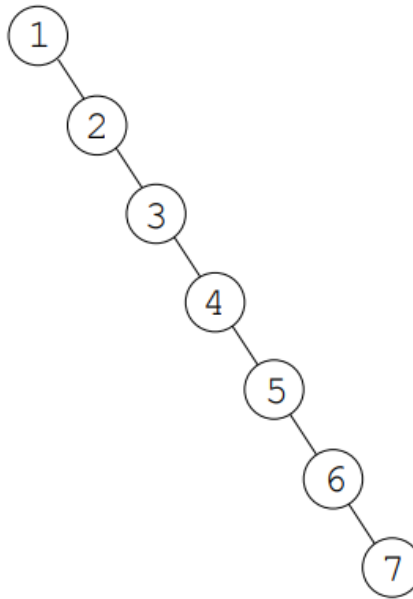
- ▶ complexidade temporal no pior caso:  $O(\log n)$





# Complexidade de Busca em Árvore Binária – Pior Caso

- ▶ após inserções, árvore binária de busca pode degenerar em uma lista



- ▶ tempo de busca pior caso:  $O(n)$





# Algoritmo de Balanceamento de Árvores Binárias – Técnica Binária de Busca

- Quando os dados chegarem, armazene-os em um array;
- Após o fluxo de dados, ordenar o array;
- Designe o para a raiz o elemento central do array;
- O array consiste agora de 2 sub-arrays: um entre o início do array e o elemento escolhido para raiz e outro entre a raiz e a extremidade do array;
- O algoritmo procede com a construção da árvore de forma recursiva.

```
void balance(T data[], int first, int last) {  
  
    if (first <= last) {  
        int middle = (first + last) / 2;  
        insert(data[middle]);  
        balance(data, first, middle-1);  
        balance(data, middle+1, last);  
    }  
}
```



# Algoritmo de Balanceamento de Árvores Binárias – Técnica Binária de Busca

Stream of data: 5 1 9 8 7 0 2 3 4 6

Array of sorted data: 0 1 2 3 4 5 6 7 8 9

(a) 0 1 2 3 4 5 6 7 8 9

4

(b) 0 1 2 3 4 5 6 7 8 9

4

1

(c) 0 1 2 3 4 5 6 7 8 9

4

1

0

2

(d) 0 1 2 3 4 5 6 7 8 9

4

1

7

0

2

5

8

3

6

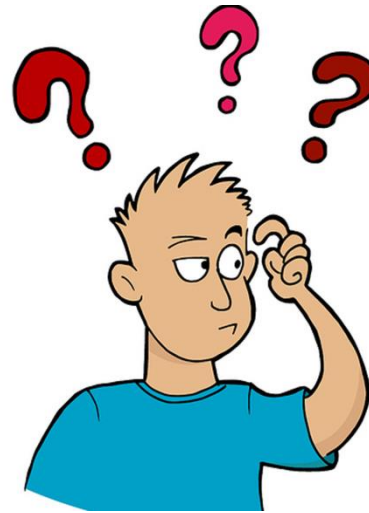
9







Hã algum problema com esse algoritmo?



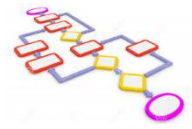


# Algoritmo de Balanceamento de Árvores Binárias – Técnica Binária de Busca

```
void balance(T data[], int first, int last) {  
  
    if (first <= last) {  
        int middle = (first + last) / 2;  
        insert(data[middle]);  
        balance(data, first, middle-1);  
        balance(data, middle+1, last);  
    }  
}
```

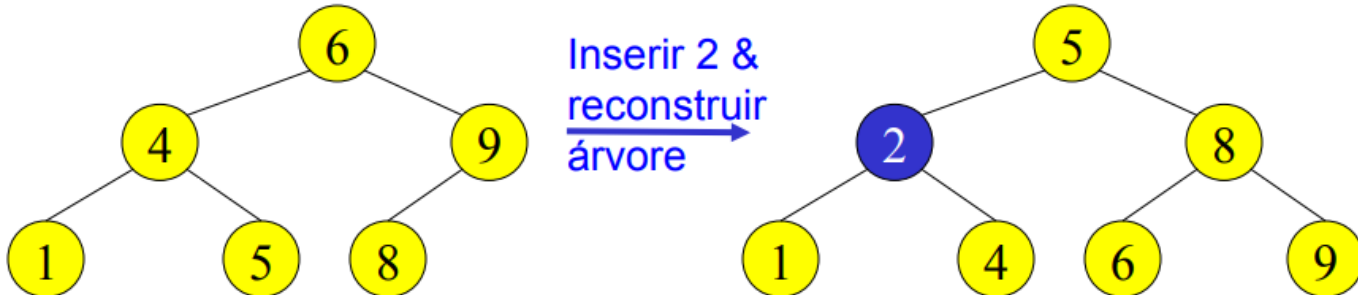
- Esse algoritmo tem um sério inconveniente;
- Todos os dados precisam ser colocados em um array antes da criação da árvore;
- Nesse caso, o algoritmo pode ser inadequado quando a árvore tem que ser usada enquanto os dados a serem incluídos na árvore ainda estiverem chegando.





## Balanceamento in locus

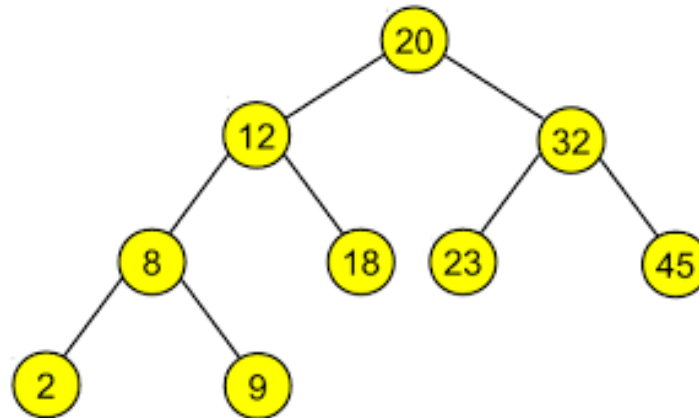
- Queremos uma **árvore completa** após cada operação
  - › a árvore está cheia exceto possivelmente na parte inferior direita.
- Isso é caro computacionalmente
  - › Por exemplo, inserir 2 na árvore da esquerda e então reconstruir toda a árvore.





# Árvore AVL

- ✓ É uma árvore de busca binária de altura autobalanceada ou de altura equilibrada;
- ✓ Em tal árvore, as **alturas** das suas sub-árvores a partir de cada nó diferem de no máximo 1 unidade;
- ✓ O nome **AVL** vem de seus criadores (Adelson Velsky e Landis)





# Árvore AVL

- ✓ Em virtude do auto-balanceamento da árvore, as operações de busca, inserção e remoção em uma árvore com  $n$  elementos podem ser feitas, mesmo no pior caso, em  **$O(\log n)$** ;
- ✓ Um teorema demonstrado por Adelson-Velskii e Landis, garante que a árvore balanceada nunca será **45%** mas alta que a correspondente árvore perfeitamente balanceada, independente do número de nós existentes;

