BC1424
Algoritmos e Estruturas de Dados I

Aula 16:

Árvores (introdução)

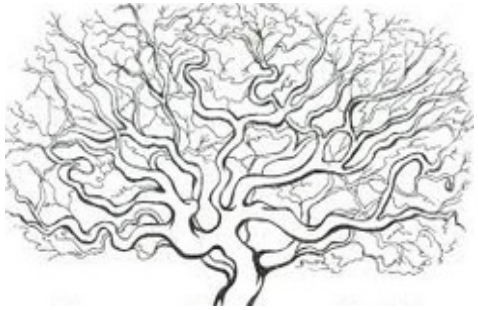
Prof. Jesús P. Mena-Chalco
jesus.mena@ufabc.edu.br

1Q-2015

Árvores

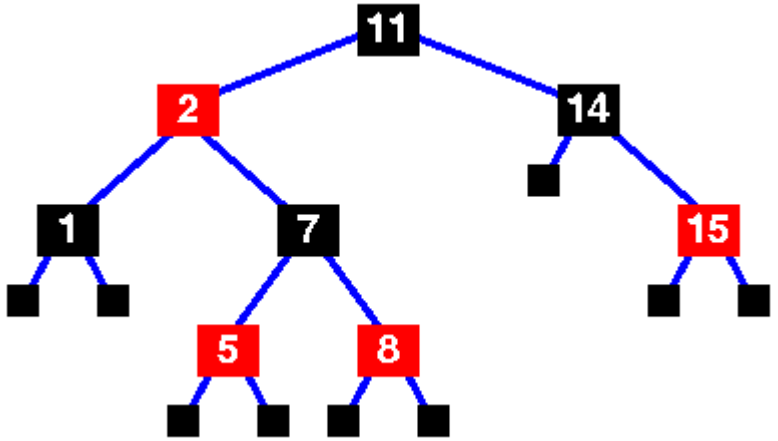
- Uma árvore é uma estrutura de dados **mais geral que uma lista ligada**.

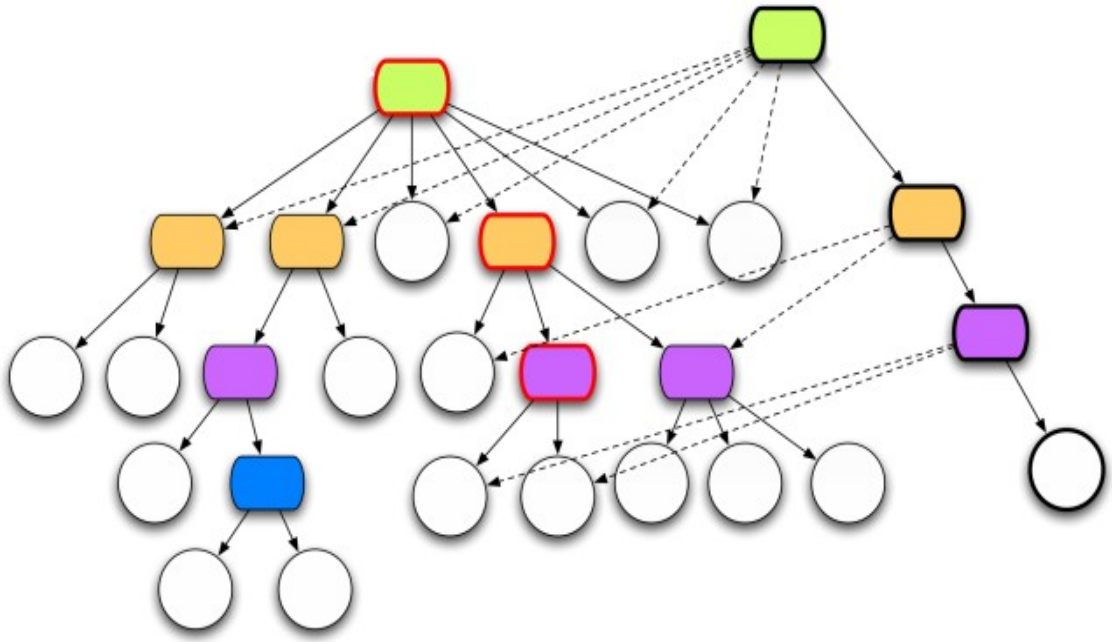
- Nessa aula examinaremos os conceitos e operações “mais simples” sobre árvores binárias.

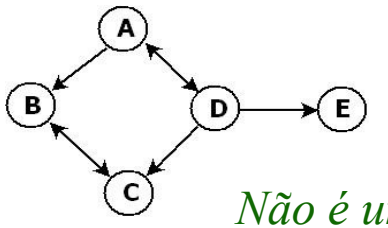


Árvores

- São estruturas não lineares.
- Representação natural para dados aninhados.
- Muito úteis para resolver uma enorme variedade do problema envolvendo algoritmos.

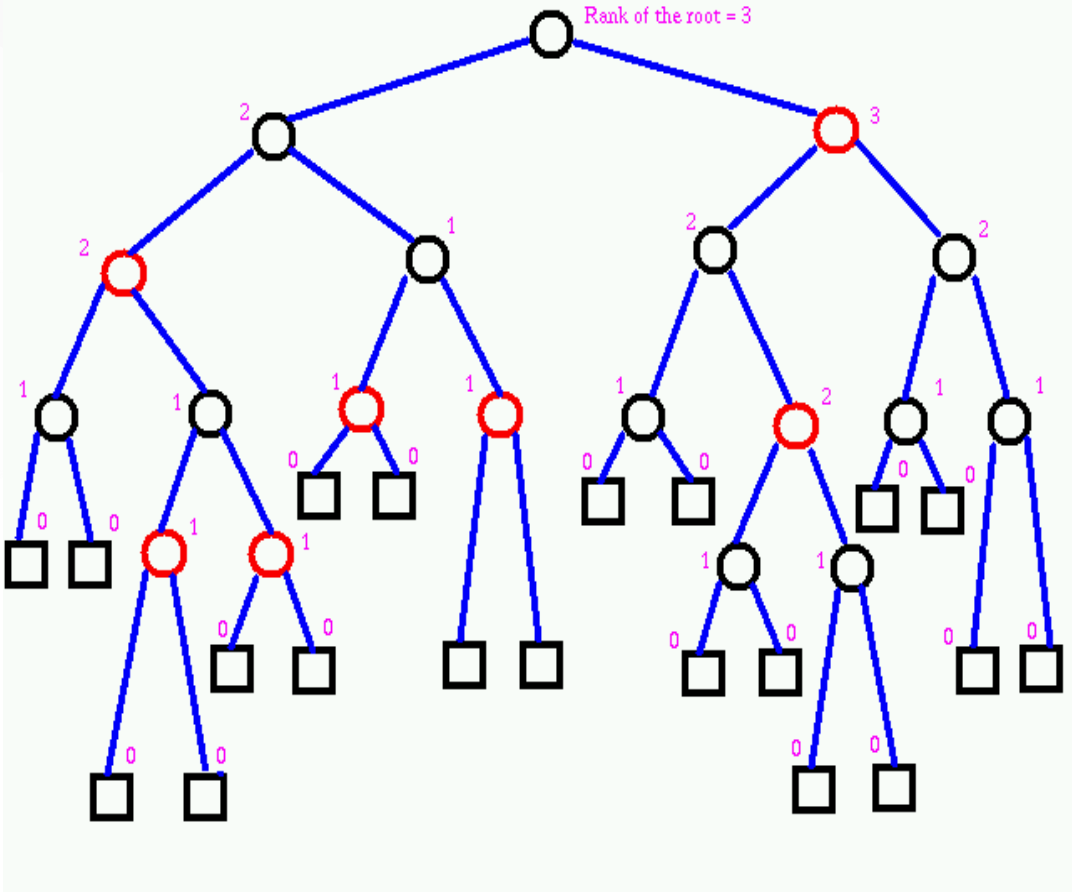


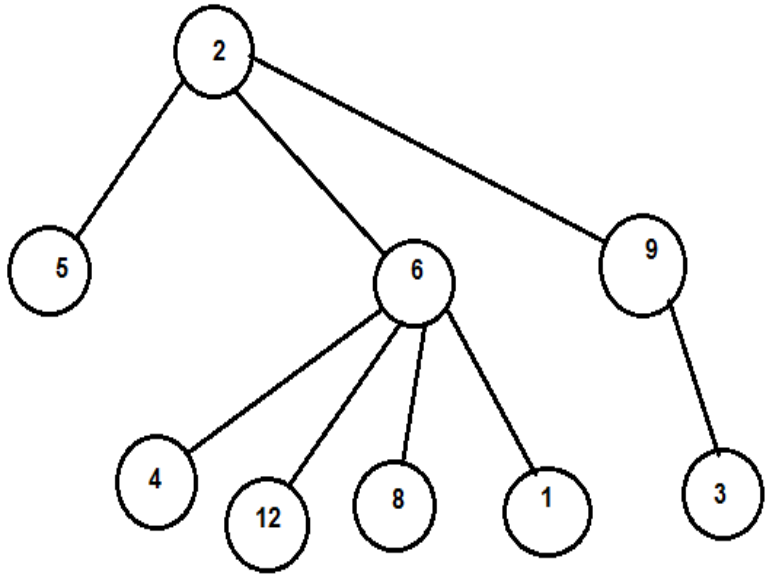




Não é uma árvore

Árvores





Árvores de pesquisa

A árvore de pesquisa é uma **estrutura muito eficiente para armazenar informação**.

Apropriada quando existe necessidade de considerar todos ou alguma combinação de:

- **Acesso direto e sequencial eficientes.**
- **Facilidade de inserção e retirada de elementos.**
- **Boa taxa de utilização de memória.**
- **Utilização de memória principal e secundária.**

Árvores binárias

Nós e filhos

Uma árvore binária (=binary trees) é um conjunto de registros que satisfaz certas condições.

Nós e filhos

Uma árvore binária (=binary trees) é um conjunto de registros que satisfaz certas condições.

Os registros serão chamados de **nós (**poderiam também ser chamadas de **células)**

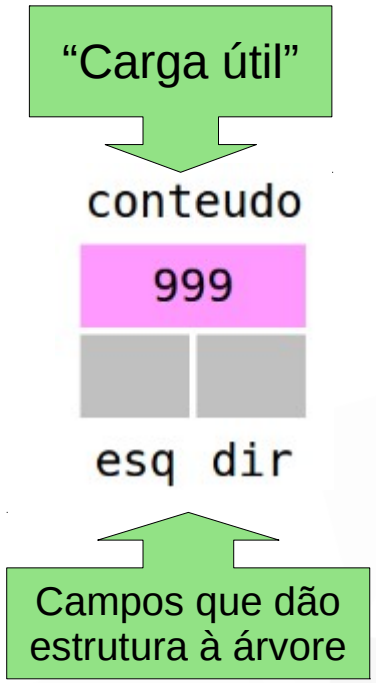
- Cada nó tem um endereço.
- Suporemos que por enquanto cada nó tem 3 campos:
  - Um número inteiro.
  - Dois ponteiros para nós.

Nós e filhos

```
struct cel {
    int conteudo;
    struct cel *esq;
    struct cel *dir;
};
```

```
typedef struct cel no;
```

```
typedef no *arvore;
```



→ *O nó folha (=leaf) é um nó que não tem filho algum.*
→ *Se x tiver um pai, essa árvore é uma subárvore de alguma árvore maior.*

Varredura

Varredura

Muitos algoritmos sobre árvores ficam mais simples quando escritos de forma recursiva.

Varredura

Ao contrário de uma lista encadeada, **uma árvore binária pode ser percorrida de muitas maneiras diferentes**.

Uma maneira particularmente importante é a ordem **esquerda-raiz-direita (e-r-d)**

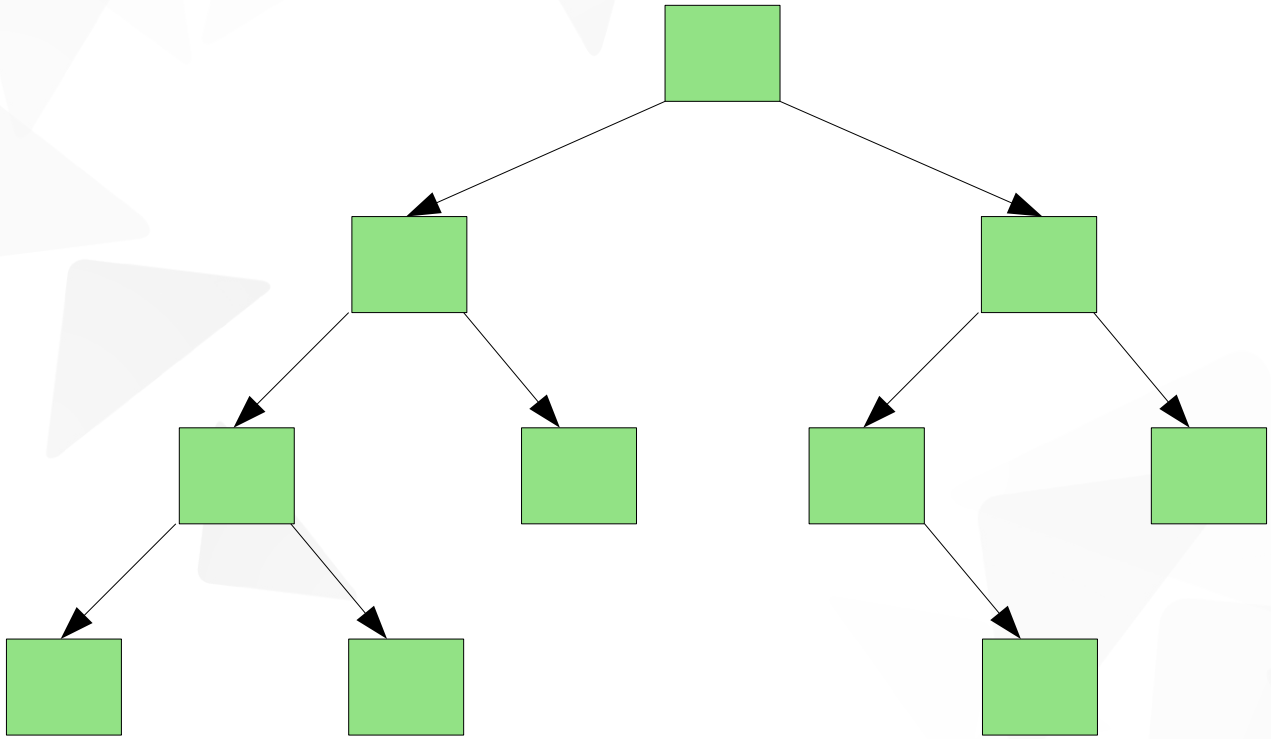
→ **inorder traversal**

→ **percurso em inorder**

- A subárvore esquerda da raiz, **em ordem e-r-d**;
- Depois a raiz;
- Depois a subárvore direita da raiz, **em ordem e-r-d.**

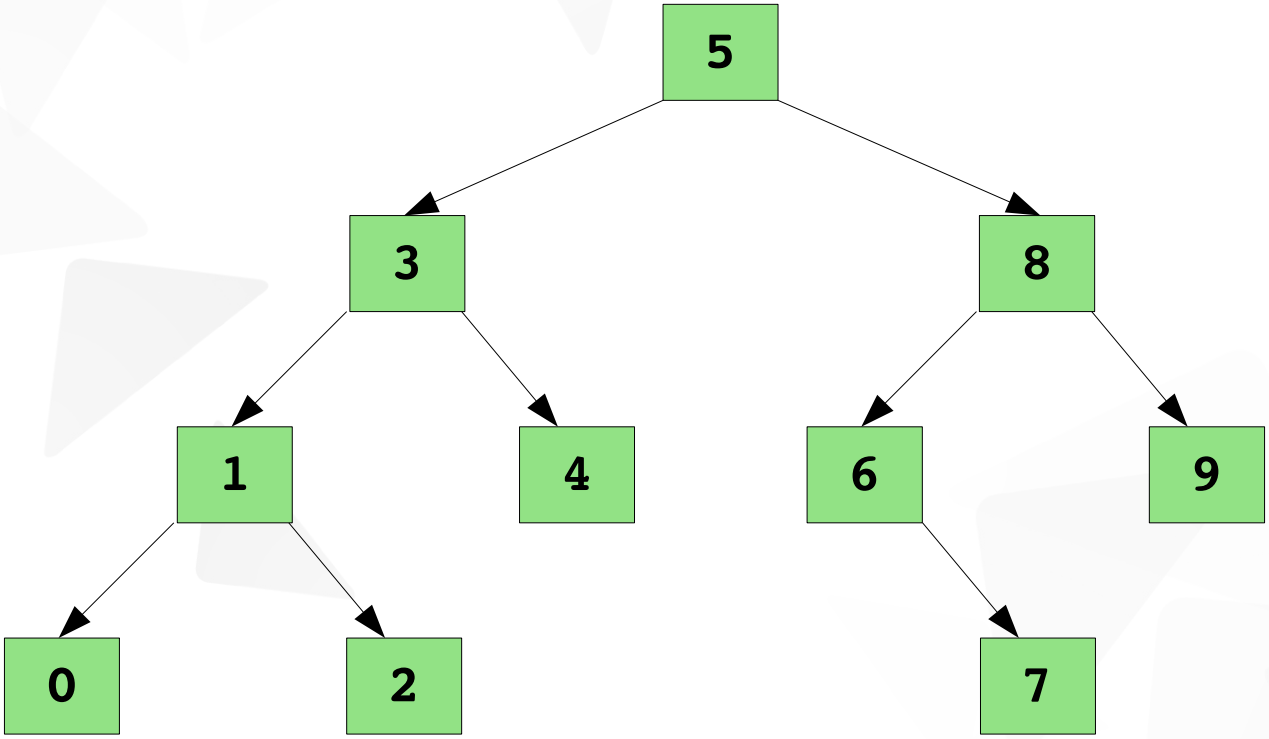
Varredura

Como seria a varredura e-r-d?



Varredura

Na figura abaixo, os nós estão numeradas na ordem da varredura e-r-d.



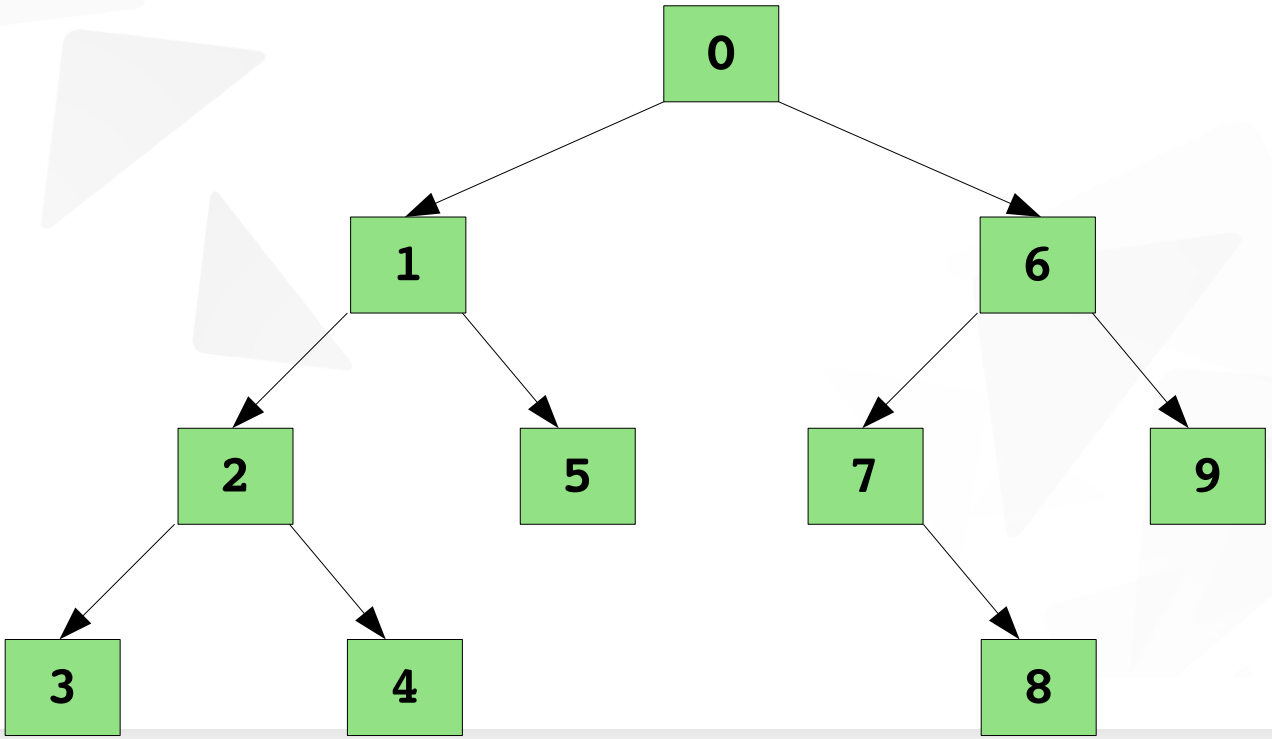
Varredura

Uma função recursiva que faz a **varredura e-r-d** de uma **árvore binária r**:

```
void erd (no *r) {
    if (r!=NULL) {
        erd(r->esq);
        printf("%d\n", r->conteudo);
        erd(r->dir);
    }
}
```

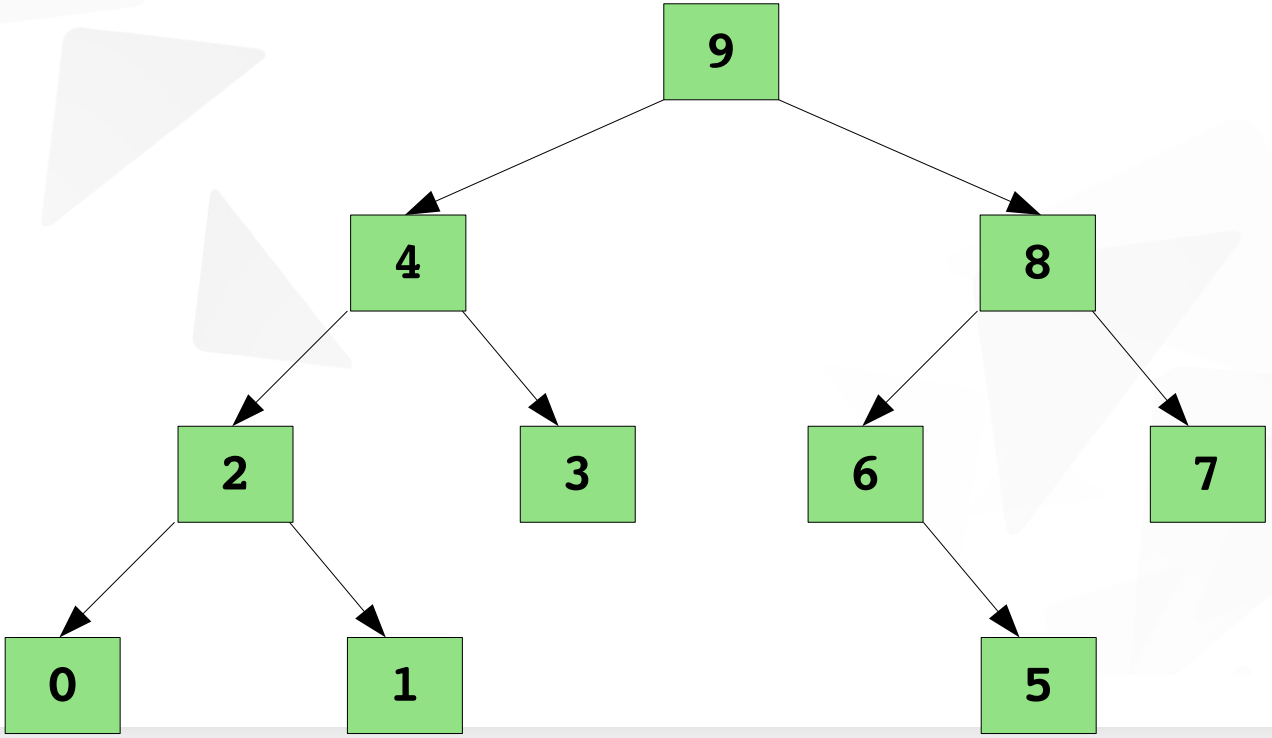
Varredura

```c
void red (no *r) {
    if (r!=NULL) {
        printf("%d\n", r->conteudo);
        erd(r->esq);
        erd(r->dir);
    }
}
```

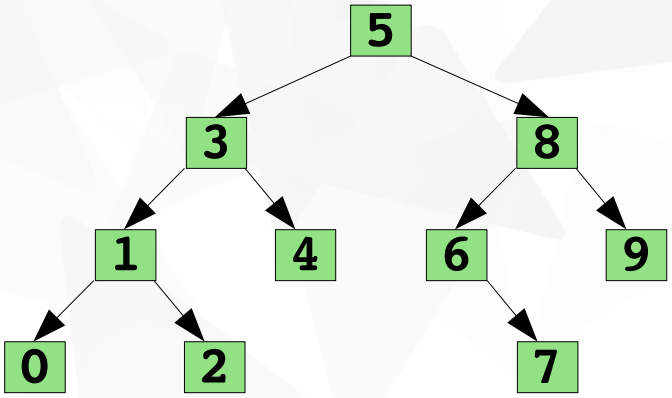


Varredura

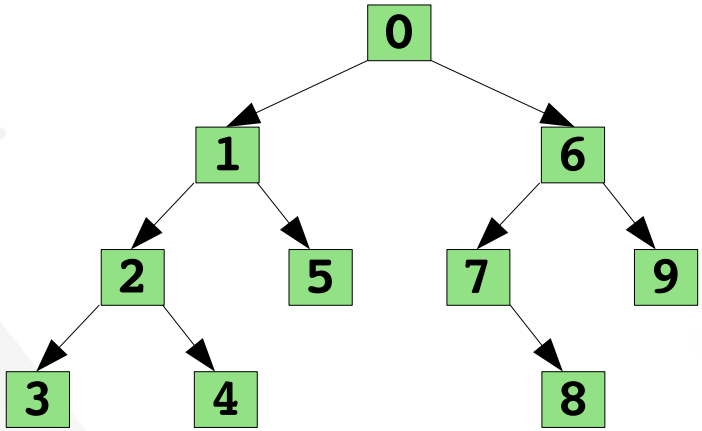
```
void edr (no *r) {
    if (r!=NULL) {
        erd(r->esq);
        erd(r->dir);
        printf("%d\n", r->conteudo);
    }
}
```



Varredura

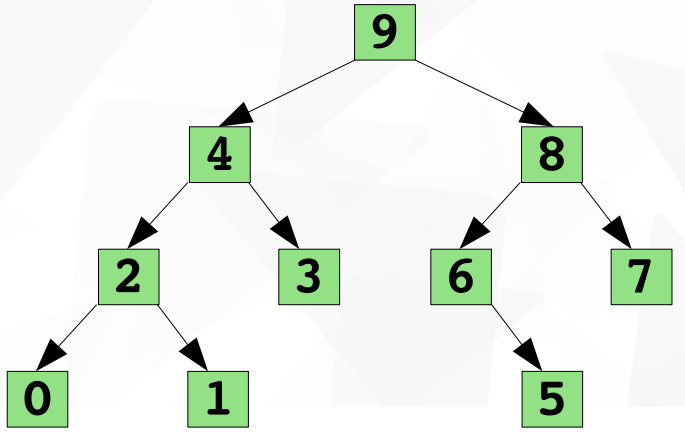


erd



red

edr



Exercícios

(1) Escreva uma função que calcule o número de nós de uma árvore binária.

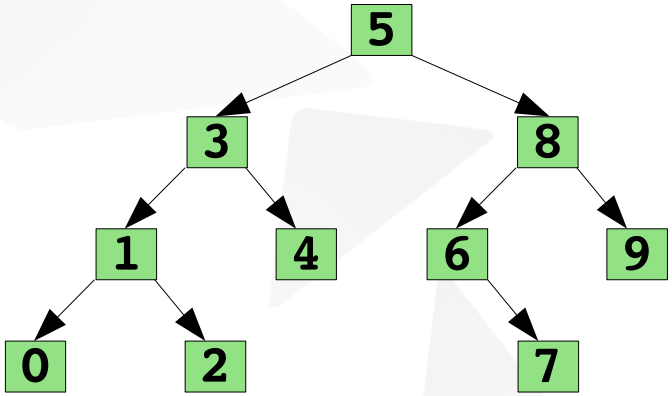
(2) Escreva uma função que imprima, em ordem e-r-d, os conteúdos das folhas de uma árvore binária.

(3) Dada uma árvore binária, encontrar um nó da árvore cujo conteúdo tenha um certo valor k.

(4) Crie uma função para encontrar o endereço do primeiro no na órdem e-r-d.

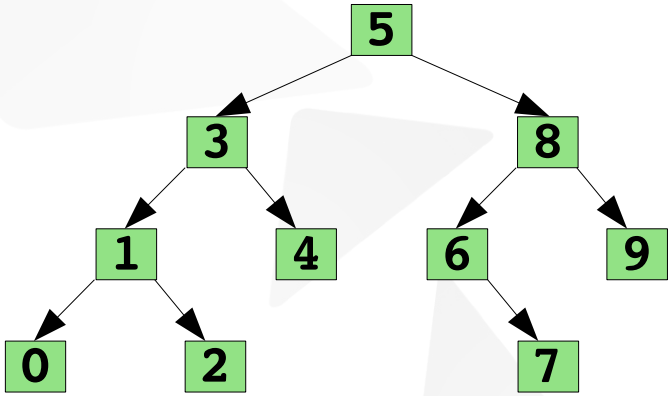
Primeiro e último nó

Crie uma função para encontrar o endereço do primeiro nó na ordem e-r-d.



Primeiro e último nó

Crie uma função para encontrar o endereço do primeiro nó na ordem e-r-d.



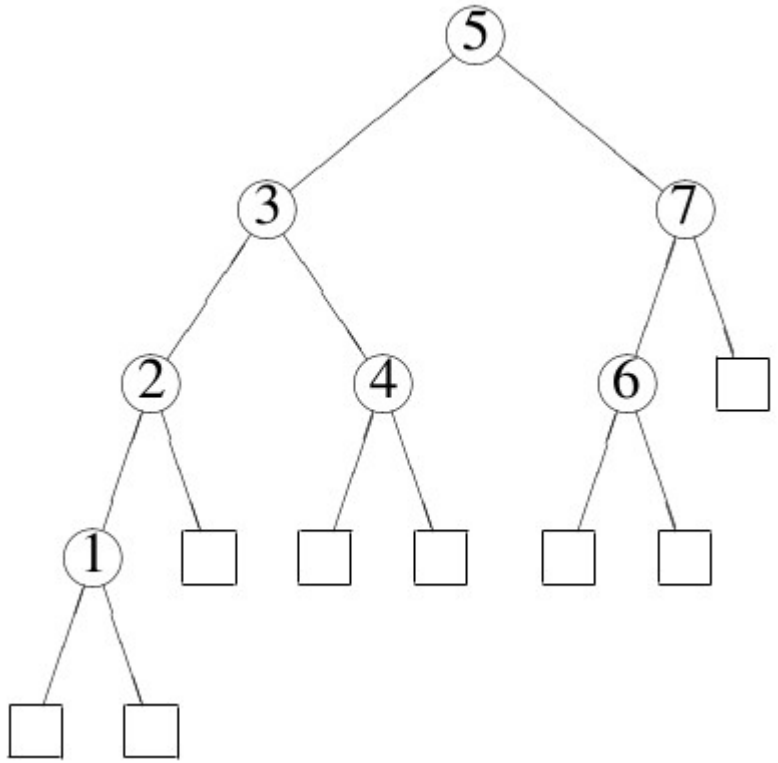

```
no* primeiroErd(no *r) {
    while (r->esq!=NULL)
        r = r->esq;
    return r;
}


no* UltimoErd(no *r) {
    while (r->dir!=NULL)
        r = r->dir;
    return r;
}
```

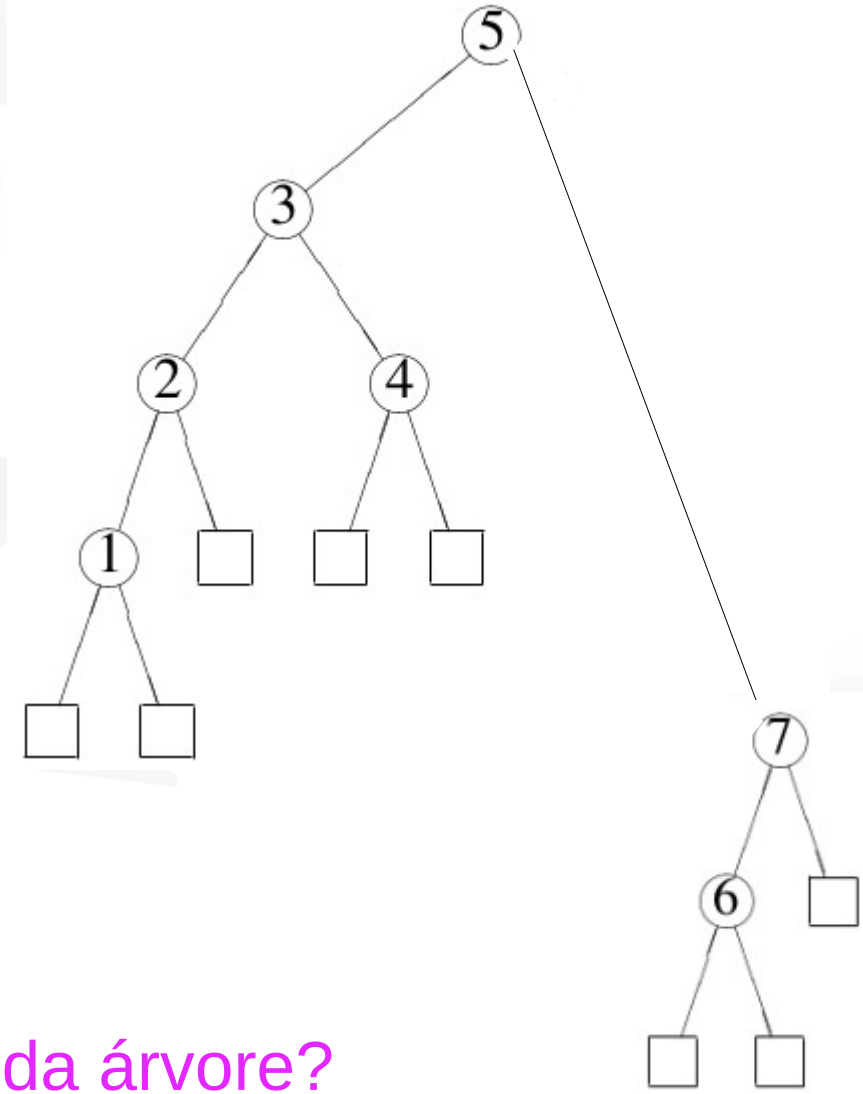
Altura de uma árvore

Altura

- O nível do nó raiz é 0.
- A altura de um nó é o comprimento do caminho mais longo deste até o nó folha.
- A altura de uma árvore é a altura do nó raiz.

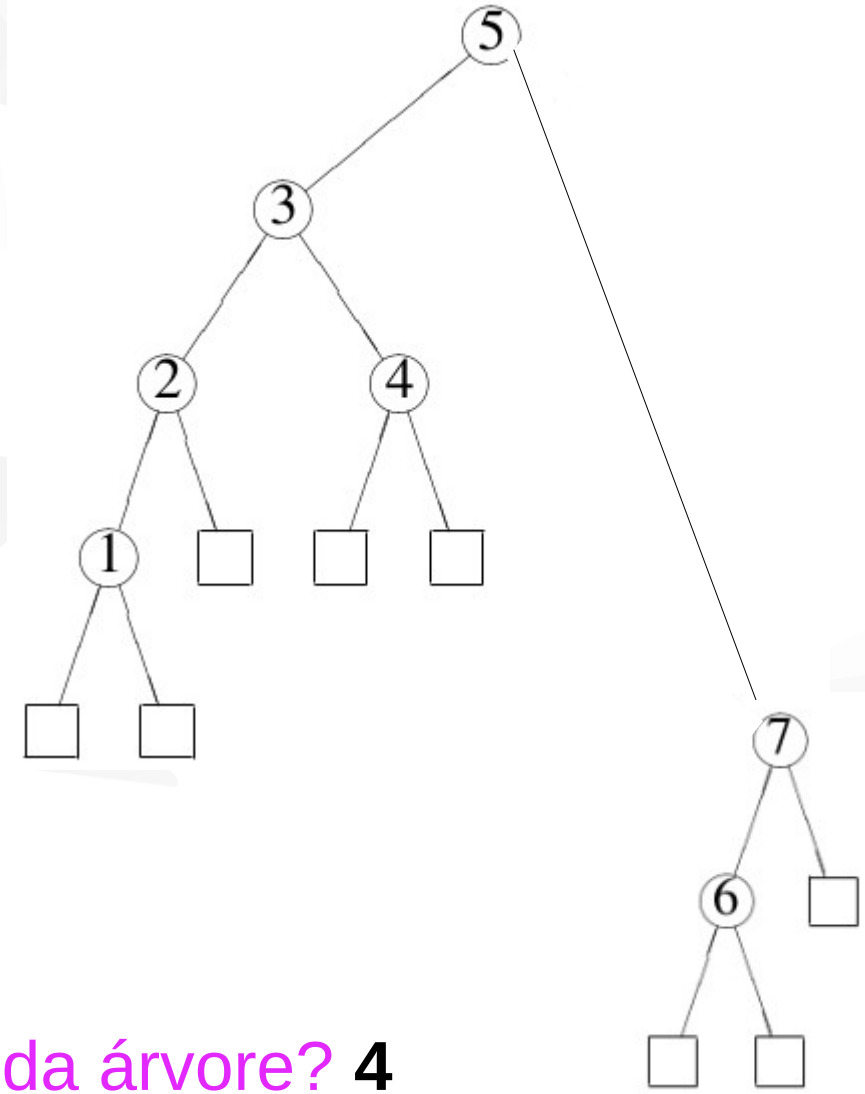


Árvores de pesquisa



Qual é a altura da árvore?

Árvores de pesquisa



Qual é a altura da árvore? **4**

Altura de uma árvore

A altura de um nó **x** em uma **árvore binária** é a distância entre x e o seu descendente mais afastado.

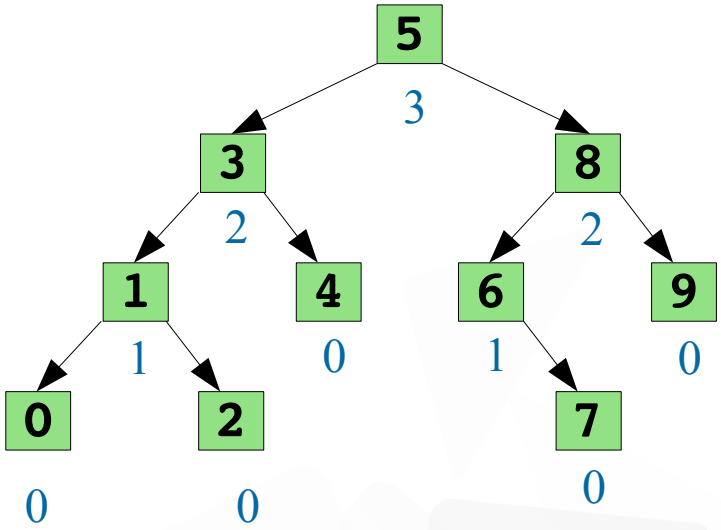
- A altura de uma árvore é a altura da raiz da árvore.
- Uma árvore com um único nó tem altura 0.

Crie uma função que calcule a altura de uma árvore.

A altura de uma árvore vazia é -1

Altura de uma árvore

```
int altura (no *r) {
    if (r==NULL)
        return -1;
    else {
        int he = altura(r->esq);
        int hd = altura(r->dir);
        if (he<hd)
            return hd+1;
        else
            return he+1;
    }
}
```

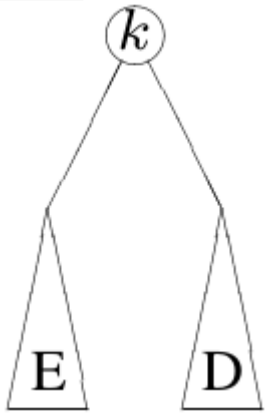


P2: 14.3.3: Árvores AVL: Uma árvores é balanceada no sentido AVL se, para cada nó x, as alturas das subárvores esquerda e direita de x diferem em no máximo uma unidade. Escreva uma função que decida se uma dada árvore é balanceada no sentido AVL. Procure escrever sua função de modo que ela visita cada nó no máximo uma vez.

Árvores binária de busca
Árvores de busca binária

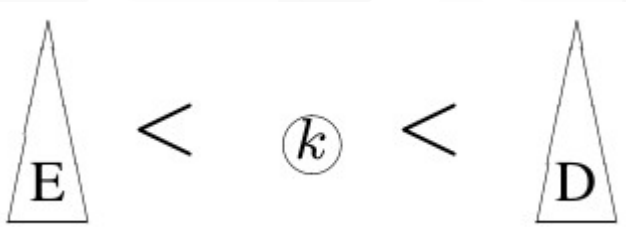
Árvores de busca binária

Para qualquer nó que contenha um registro:



Temos a relação invariante:

1. Todos os registros com chaves menores estão na sub-árvore à esquerda.
2. Todos os registros com chaves maiores estão na sub-árvore à direita.

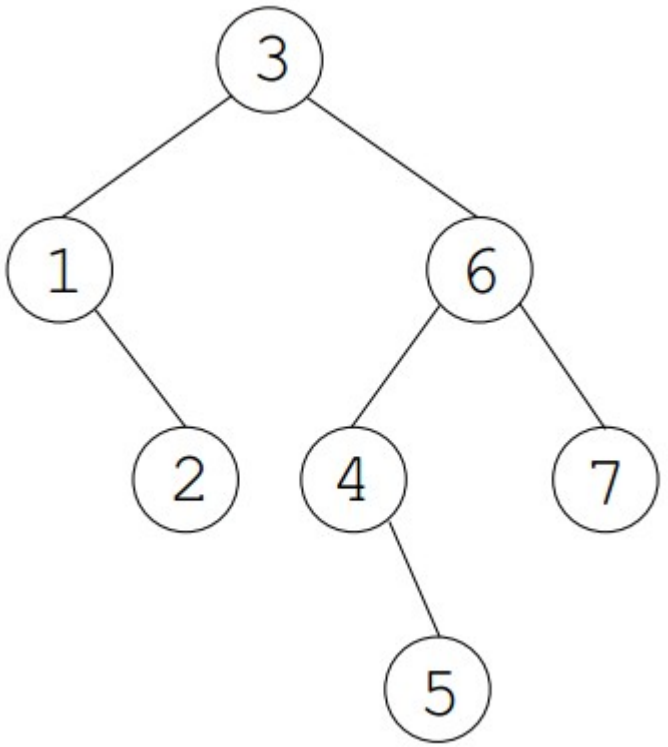


Árvores de Busca Binária

- As ABB permitem minimizar o tempo de acesso no pior caso.
- Para cada chave, separe as restantes em **maiores** ou **menores**.

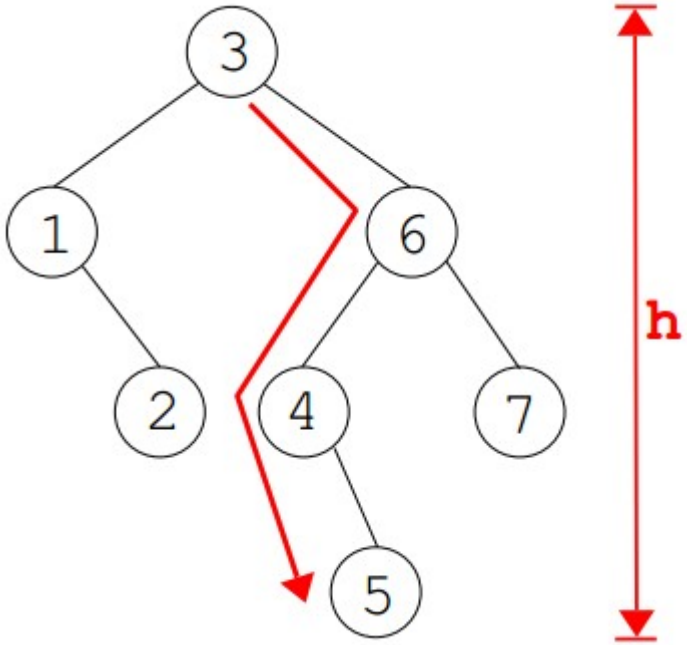
Árvores de Busca Binária

- As ABB permitem minimizar o tempo de acesso no pior caso.
- Para cada chave, separe as restantes em **maiores** ou **menores**.
- A estrutura hierárquica com divisão binária: **uma árvore binária**.



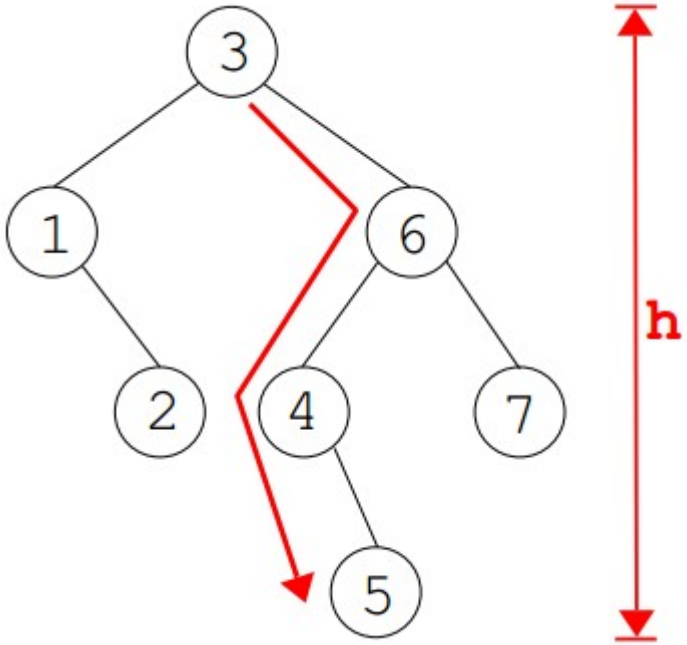
Complexidade de busca em uma ABB

- Busca em ABB = **caminho da raiz até a chave desejada**
  (ou até a folha, caso a chave não exista)



Complexidade de busca em uma ABB

- Busca em ABB = **caminho da raiz até a chave desejada**
  (ou até a folha, caso a chave não exista)

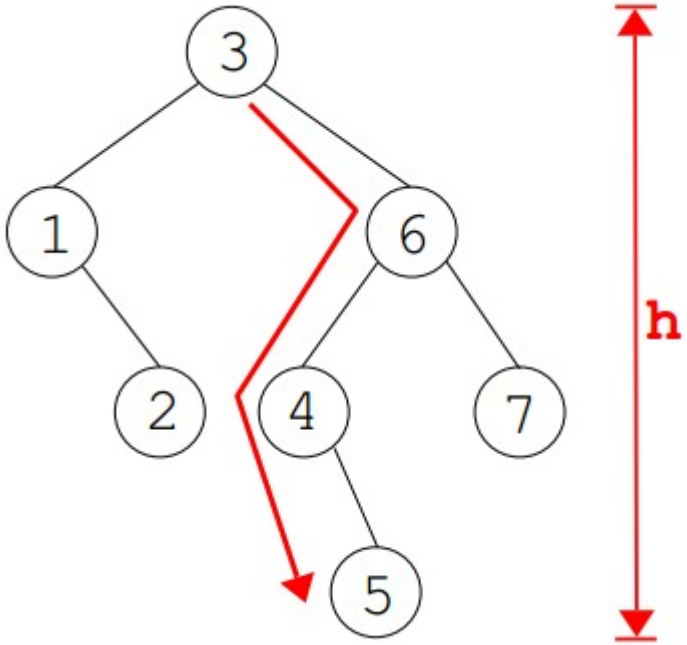


Pior caso:
Maior caminho até a folha = altura da árvore

Complexidade: $O(h)$

Complexidade de busca em uma ABB

- Busca em ABB = **caminho da raiz até a chave desejada**
  (ou até a folha, caso a chave não exista)

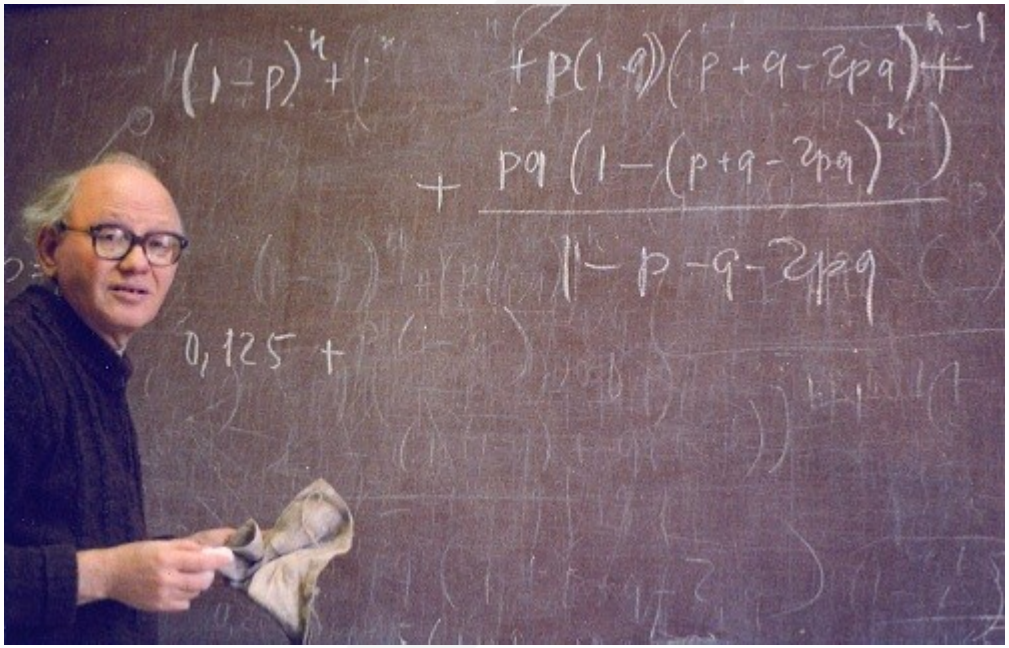


Pior caso:
Maior caminho até a folha = altura da árvore

Complexidade: $O(h)$

Uma árvore binária balanceada é aquela com altura $O(\lg n)$

AVL



Georgy M. **Adelson-Velsky**
Russia

(1922-2014/abril/26)

Evgenii Mikhailovich **Landis**
Ucrania

(1921-1997)

AVL

G.M. Adelson-Velskii y E.M. Landis
"**An algorithm for the organization of information**".
Proceedings of the USSR Academy of Sciences, vol. 146, pp. 263–266, **1962**

AN ALGORITHM FOR THE ORGANIZATION OF INFORMATION

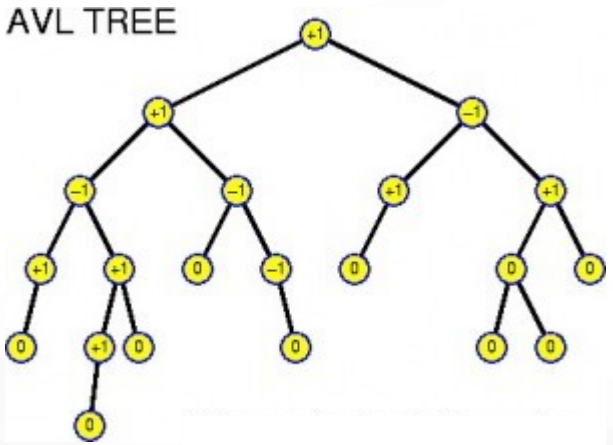
G. M. ADEL'SON-VEL'SKIĬ AND E. M. LANDIS

In the present article we discuss the organization of information contained in the cells of an automatic calculating machine. A three-address machine will be used for this study.

Statement of the problem. The information enters a machine in sequence from a certain reserve. The information element is contained in a group of cells which are arranged one after the other. A certain number (the information estimate), which is different for different elements, is contained in the information element. The information must be organized in the memory of the machine in such a way that at any moment a very large number of operations is not required to scan the information with the given evaluation and to record the new information element.

An algorithm is proposed in which both the search and the recording are carried out in $C \lg N$ operations, where N is the number of information elements which have entered at a given moment.

A part of the memory of the machine is set aside to store the incoming information. The information elements are arranged there in their order of entry. Moreover, in another part of the memory a "reference board" [1] is formed, each cell of which corresponds to one of the information elements.



AVL foi a primeira estrutura (conhecida) de árvore de altura balanceada ou altura equilibrada.

Árvores AVL

- Devido ao balanceamento da altura da árvore, as operações de:
  - Busca
  - Inserção
  - Remoção

  em uma árvore com **n** elementos podem ser efetuadas em $O(\lg n)$ mesmo no pior caso.