MC3305
Algoritmos e Estruturas de Dados II

Aula 06 – Árvores

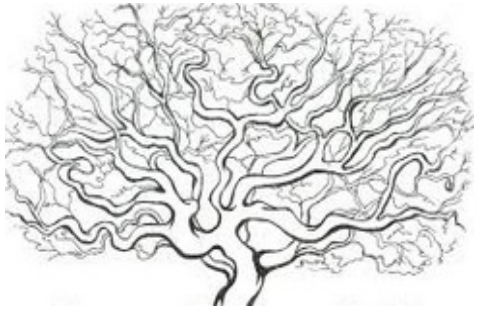
Prof. Jesús P. Mena-Chalco
jesus.mena@ufabc.edu.br

2Q-2015

Árvores

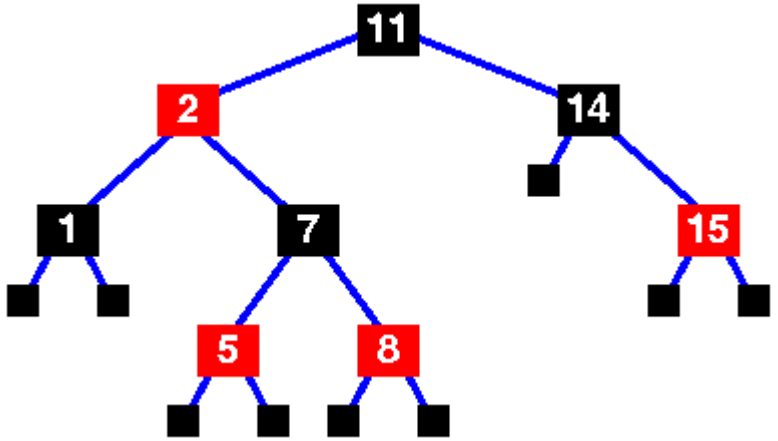
- Uma árvore é uma estrutura de dados **mais geral que uma lista ligada**.

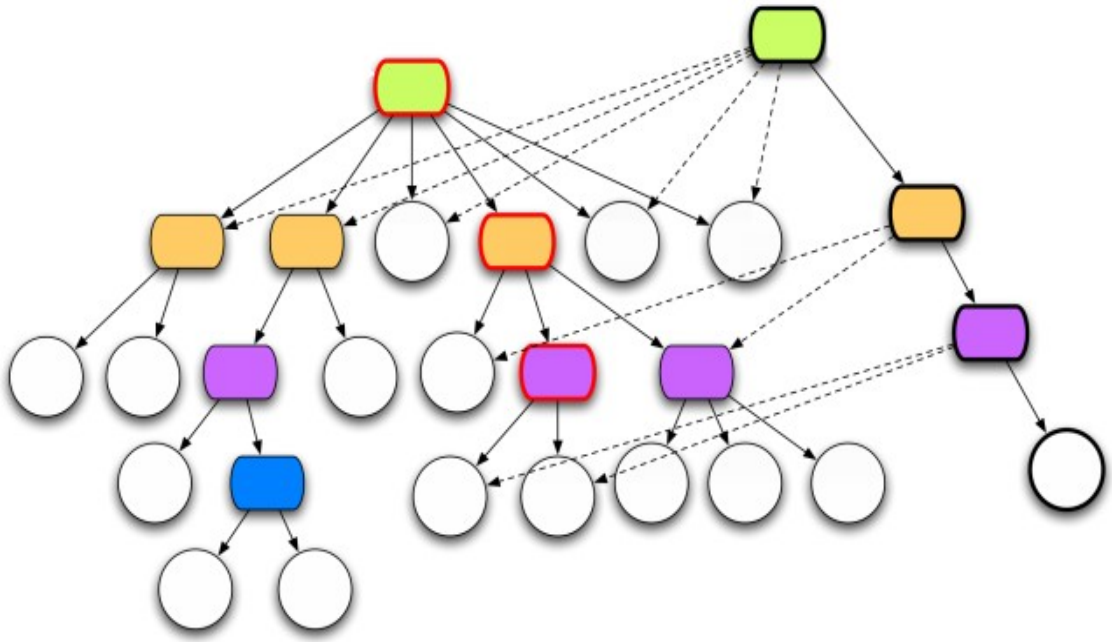
- Nessa aula examinaremos os conceitos e operações “mais simples” sobre árvores.

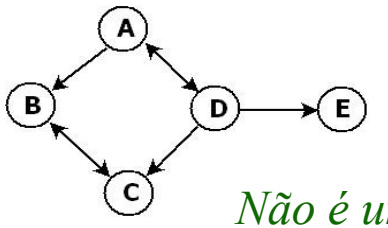


Árvores

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

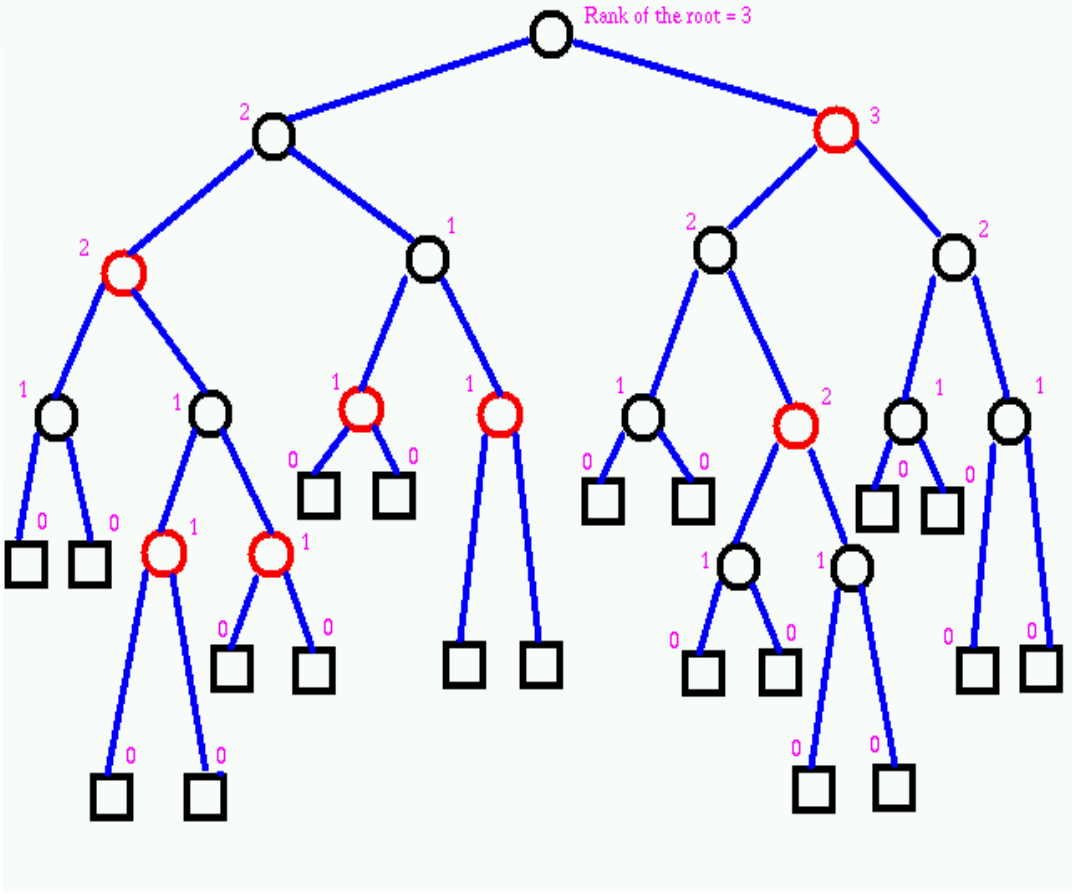


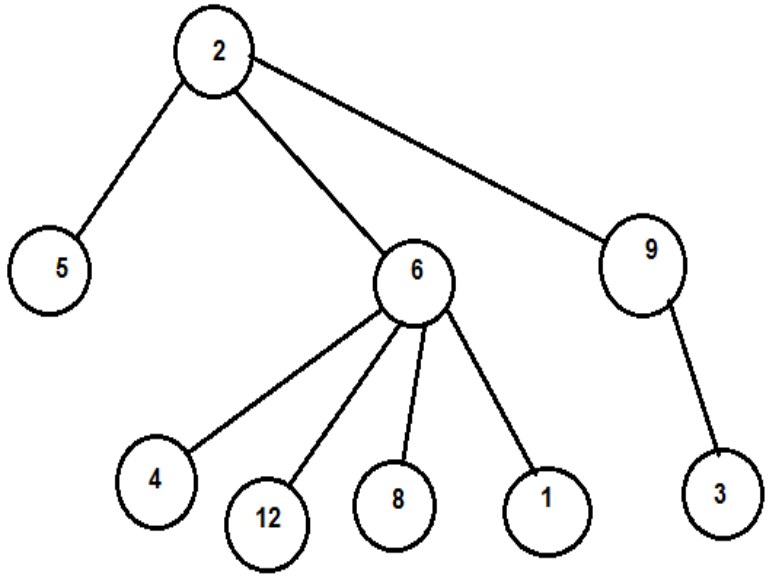




Não é uma árvore

Árvores





Árvores

Uma árvore enraizada T, é um conjunto finito de elementos denominaos nós/vértices tais que:

- T é uma árvore vazia, ou
- Existe um vértice chamado raiz de T(r(T))

Uma floresta é um conjunto de árvores.

Se v é um vértice de T, a notação T(v) indica a subárvore de T com raiz v.

- Um vértice que não possui descendentes próprios é chamado de **folha**.
- Um vértice não folha é dito **interior**.

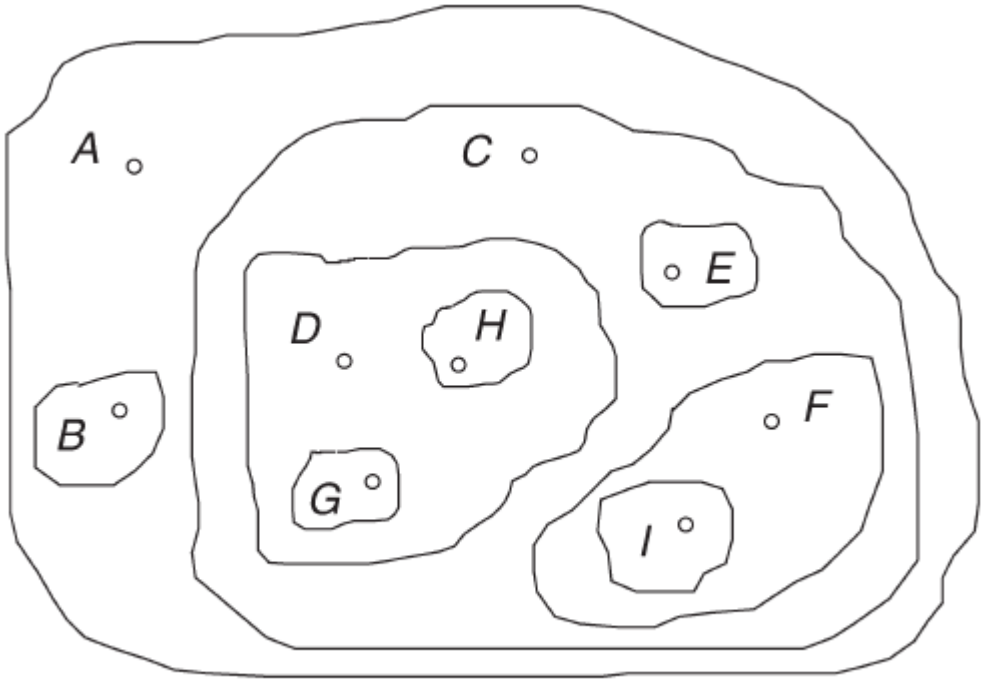


Diagrama de inclusão.

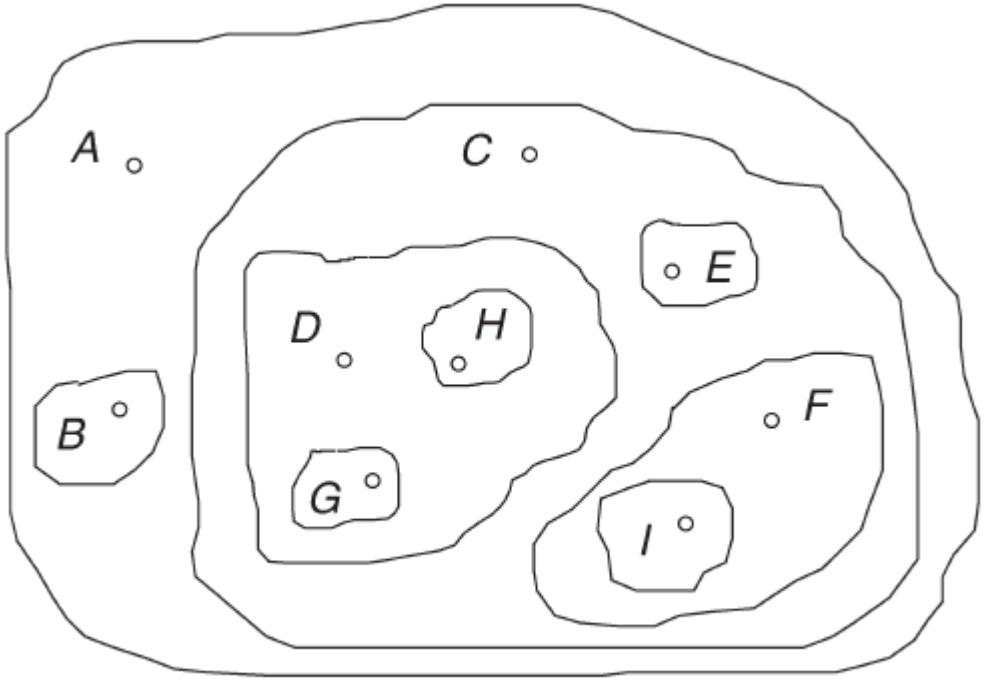
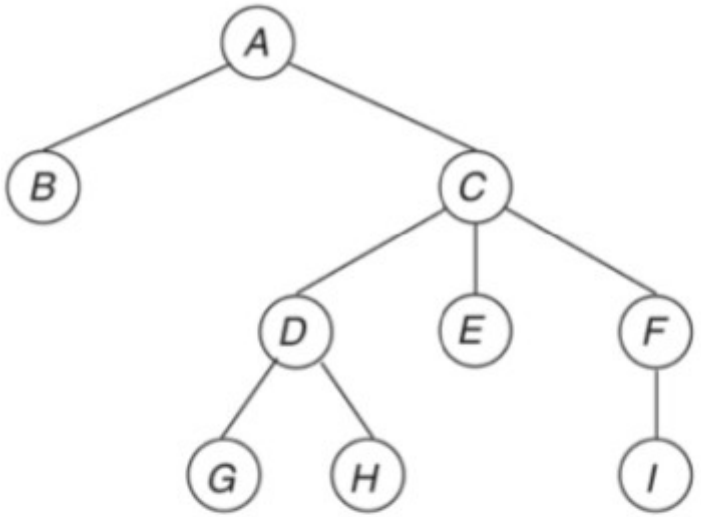
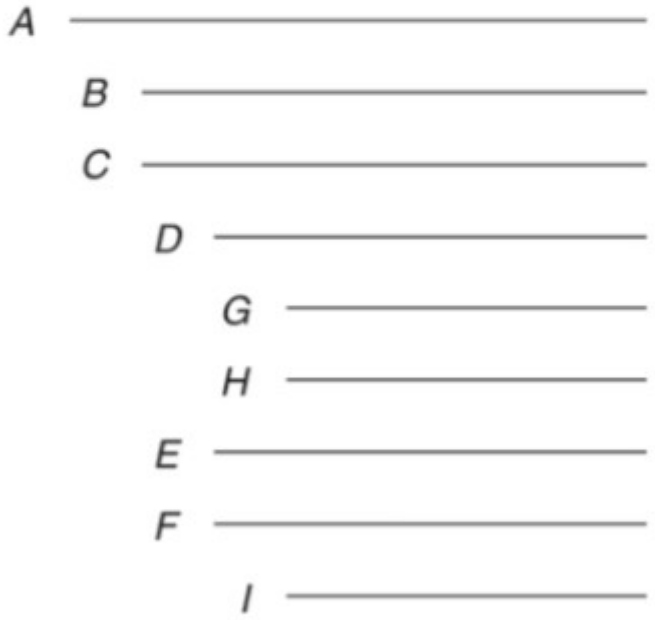


Diagrama de inclusão.

Representação hierárquica



Representação por barras



Representação por parênteses

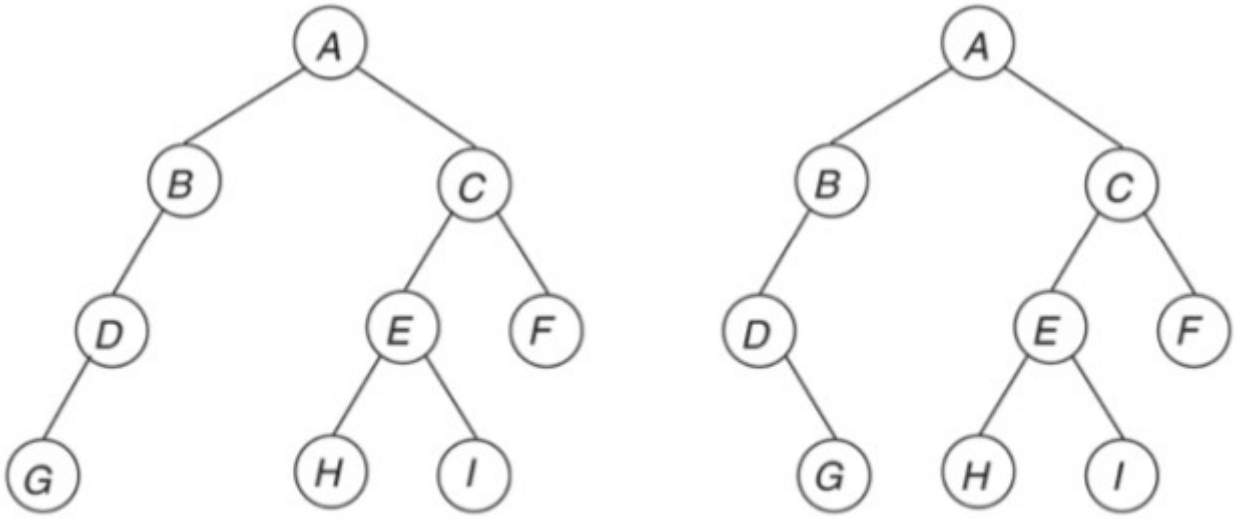
(A (B) (C (D (G) (H)) (E) (F (I))))

Árvores binárias

Árvores binárias (binary trees)

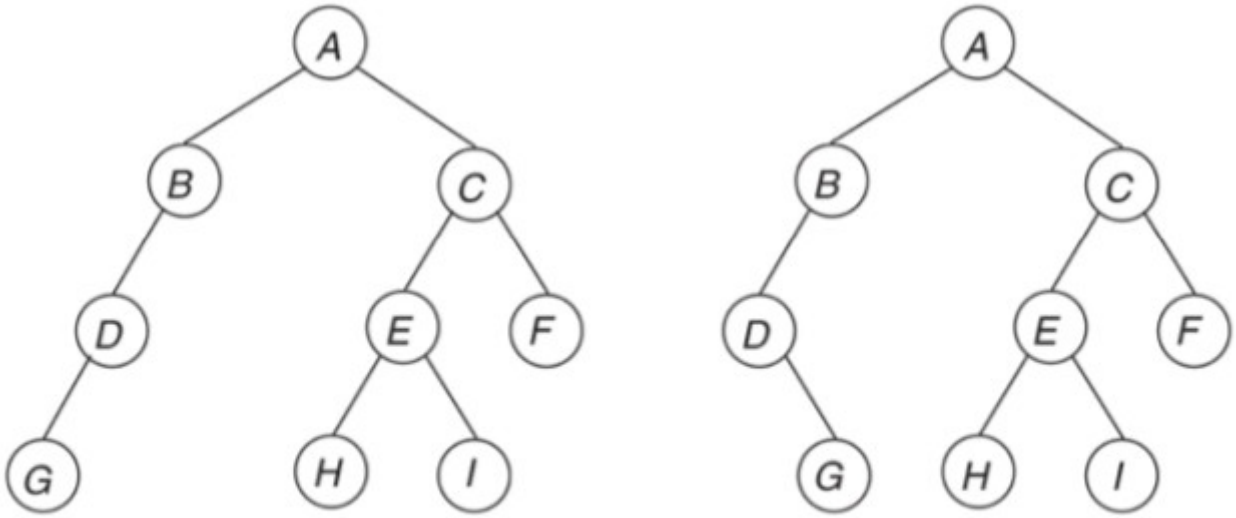
- Uma árvore binária enraizada T, é um conjunto finito de elementos denominaos nós/vértices tais que:
  - T é uma árvore vazia, ou
  - Existe um vértice chamado raiz de T(r(T)), e os restantes podem ser divididos em dois subconjuntos Te(r(T)) e Td(r(T)), a subárvore esquerda e a direita da raiz, respectivamente, as quais são também árvores binárias.

Árvores binárias (binary trees)



Árvores binárias.

Árvores binárias (binary trees)

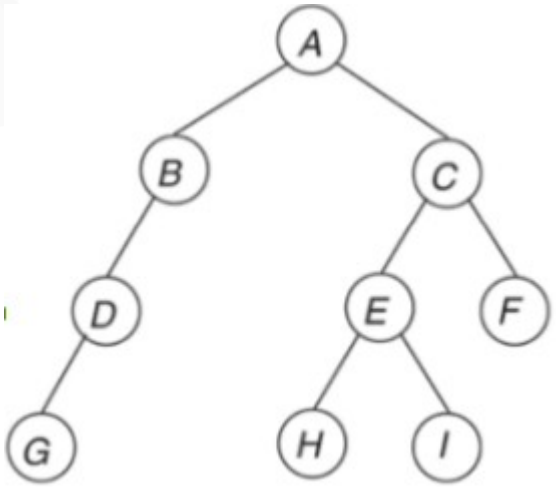


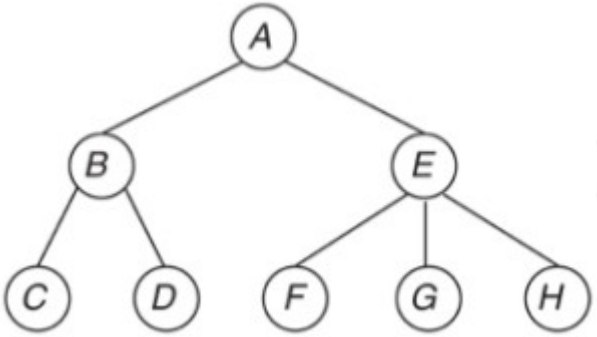
Árvores binárias.

Exemplo de Árvores Binárias, **issomórficas** (se considerdas como árvores), porém **distintas** como árvores binárias

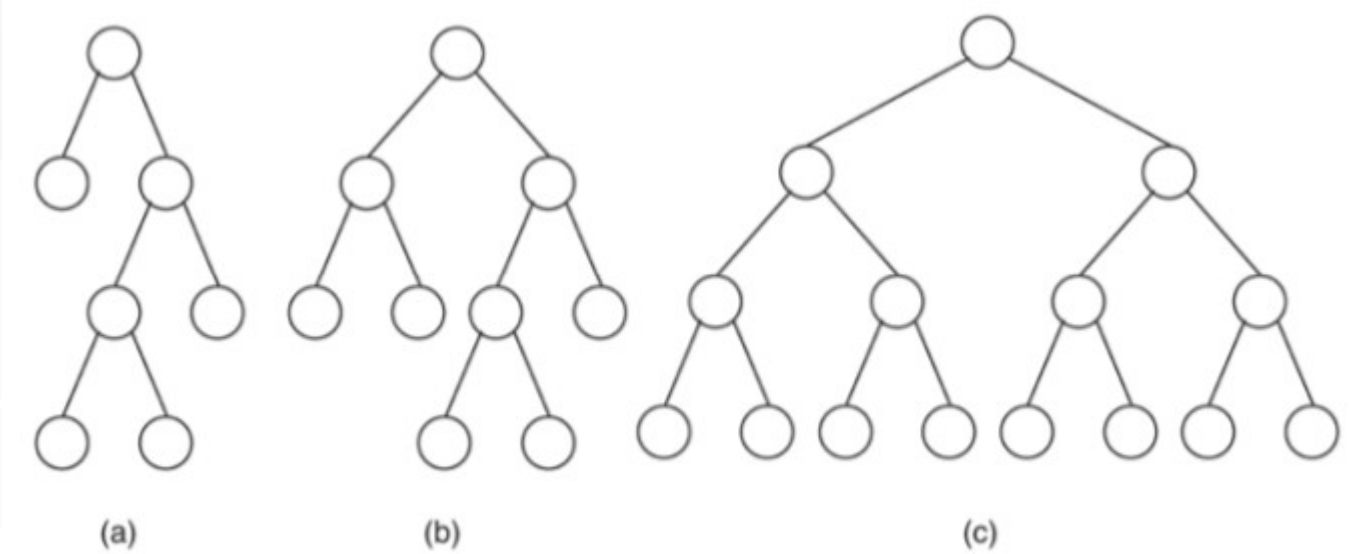
Árvores binárias (binary trees)

O número de subárvores esquerdas e direitas **vazias** em uma árvore binária com n>0 vértices é n+1.





Árvores binárias (binary trees)



Árvores estritamente binária, binária completa e cheia.

(a) Cada vértice possui 0 ou 2 filhos.

(b) Os filhos estão no último ou penúltimo nível.

(c) Os filhos estão no último nível.

Nós e filhos

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

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

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 / Percurso

→ **Visita sistematica a cada vértice**

– Uma árvore é uma estrutura não sequencial

Varredura

- Visitar um vértice significa operar, de alguma forma, com a informação a ele relativa.

- Percorrer uma árvore significa visitar os seus vértices exatamente um vez.

  Contudo, no processo de percorrer a árvore pode ser necessário passar várias vezes por alguns de seus vértices sem visitá-los.

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

Varredura

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

→ **inorder traversal.**

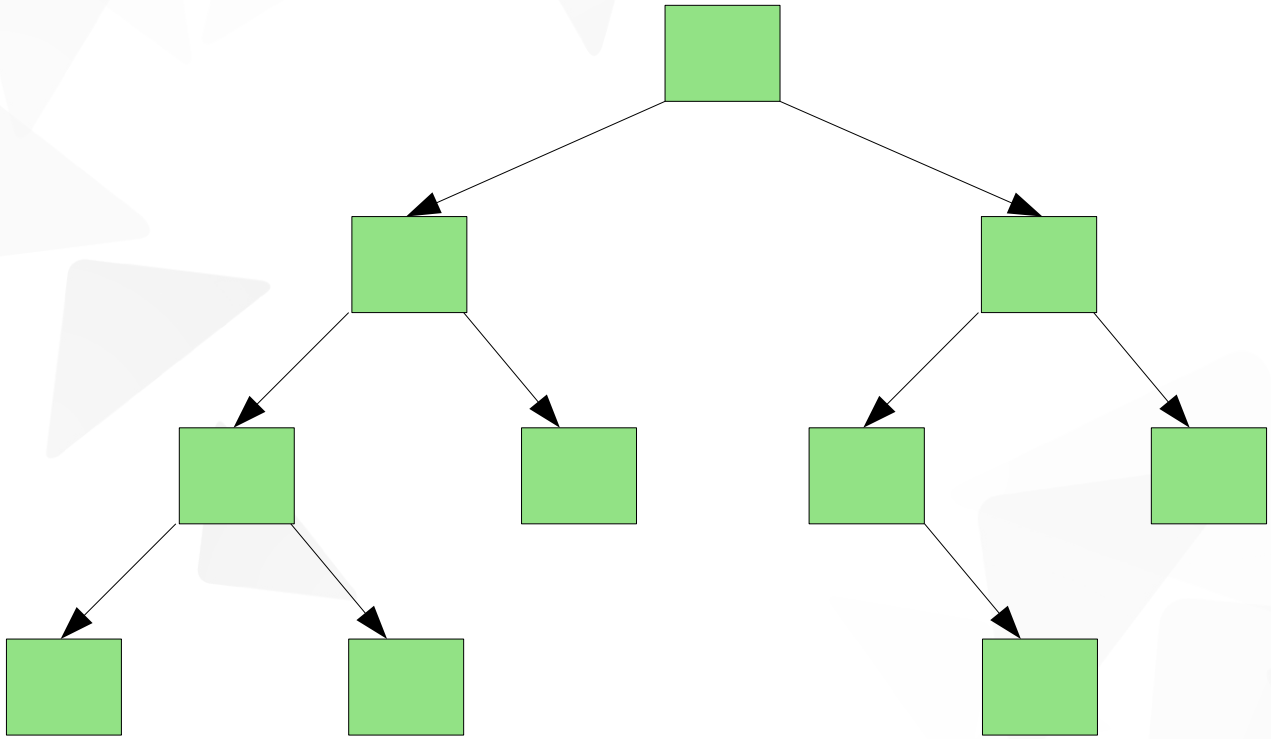
→ **percurso em ordem.**

→ **percurso em ordem simétrica.**

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

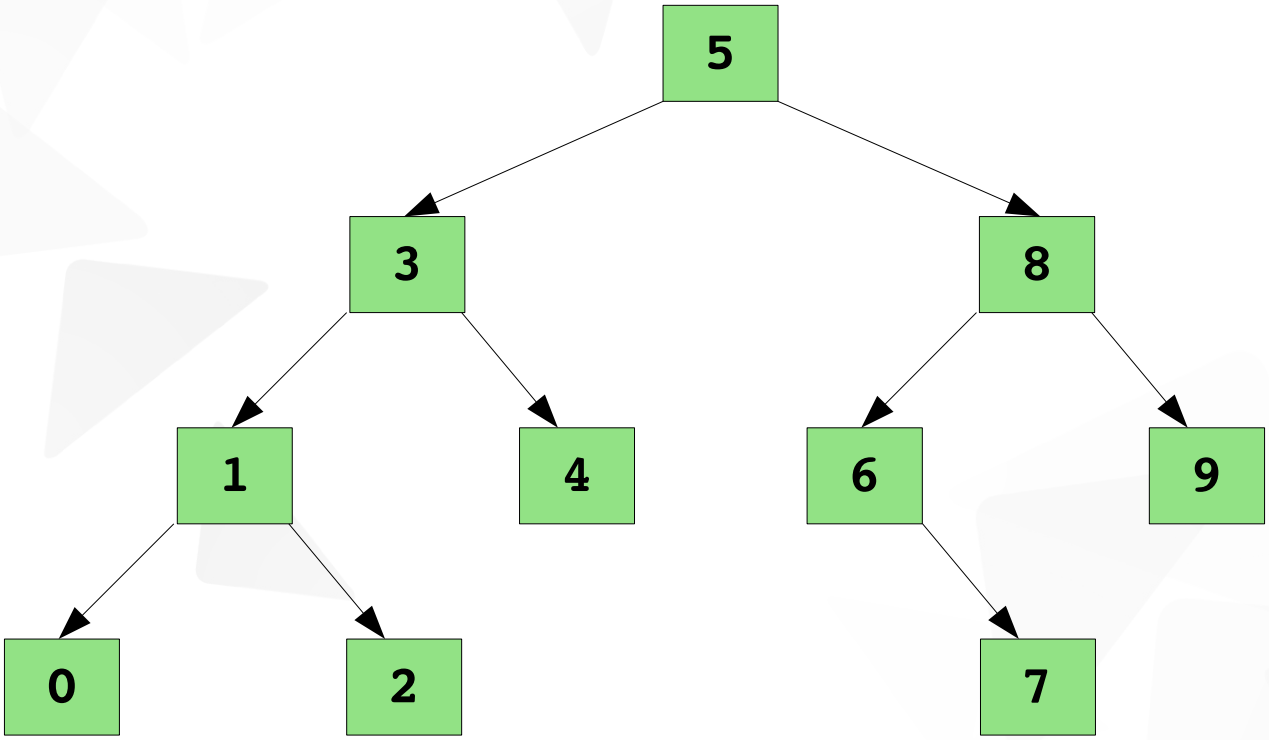
Varredura

Como seria a varredura e-r-d?



Varredura

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



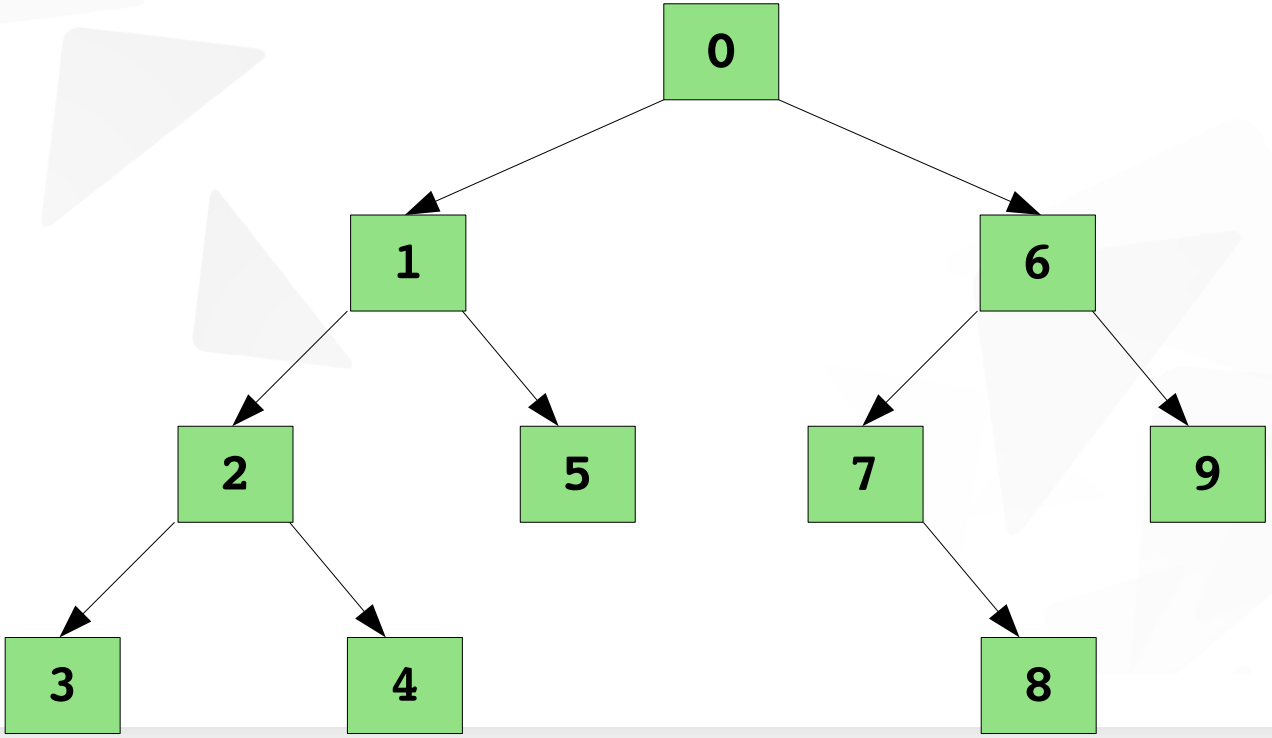
Varredura

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);
    }
}
```

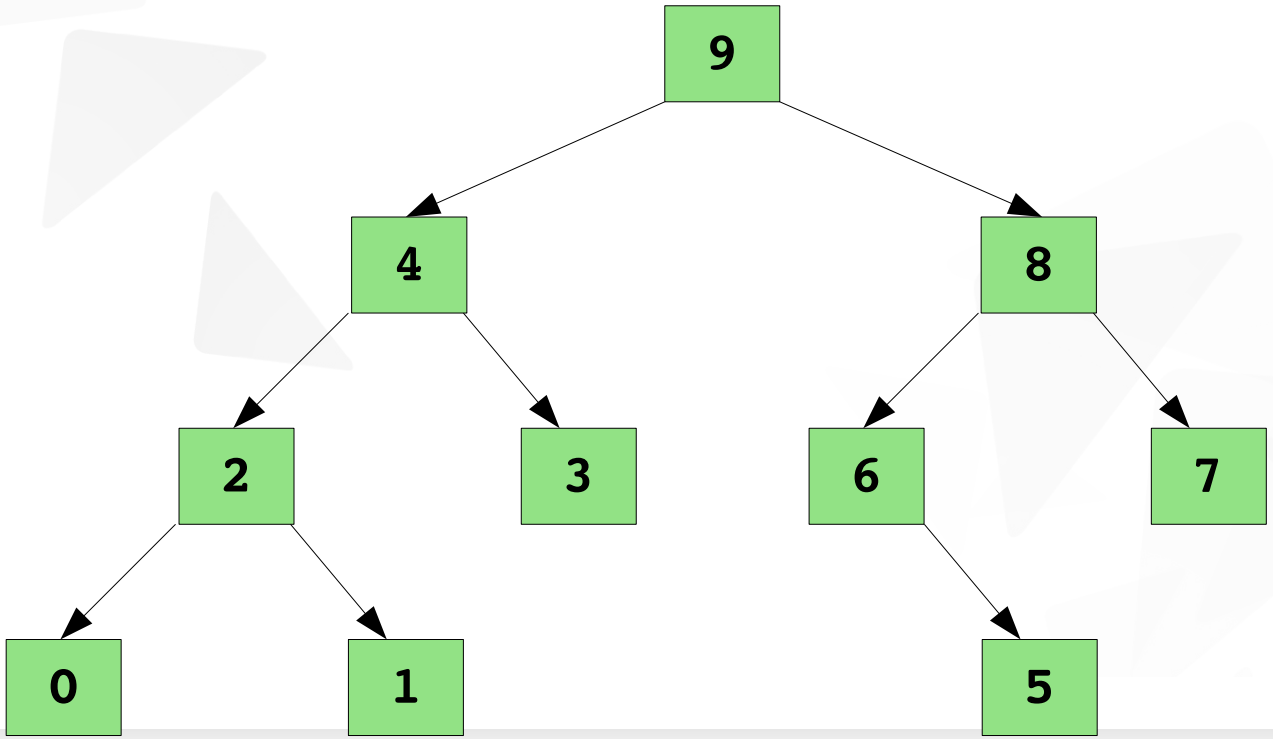
Varredura

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

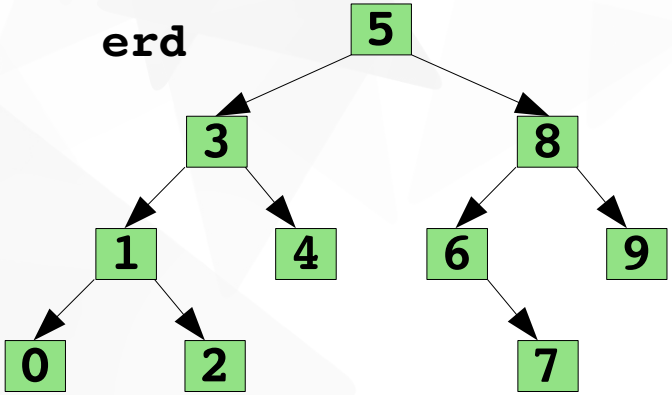


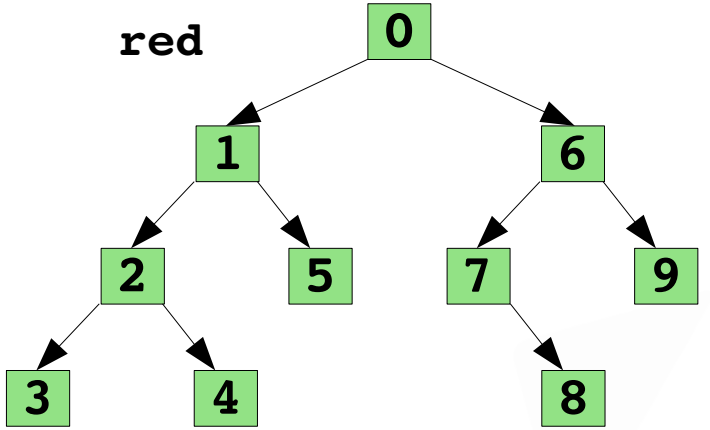
Varredura

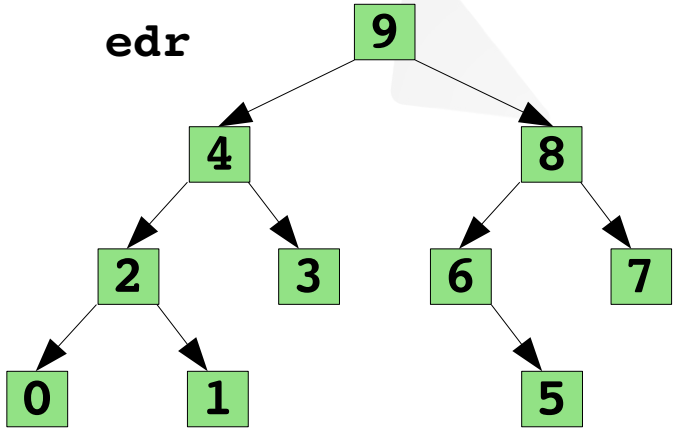
```
void edr (no *r) {
    if (r!=NULL) {
        edr(r->esq);
        edr(r->dir);
        printf("%d\n", r->conteudo);
    }
}
```



Varredura







Se n é o número de vértices, e a visita para cada vértice tiver um custo constante, a visita de todos os vértices teria um custo de *O(n)*

Varredura

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

```
// Recebe a raiz r de uma árvore binária
// Imprime os conteúdos dos vértices em ordem: erd

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

Exercício: escrever uma versão **iterativa** desta função.

Varredura

```
void erd_i (no *r) {
    int t=0;
    no *x, *p[100]; //p é uma pilha [0..t-1] elementos
    x = r;

    while (x!=NULL || t>0) {
        if (x!=NULL){
            p[t] = x;
            t++;
            x = x->esq;
        }
        else {
            t--;
            x = p[t];
            printf("%d\n", x->conteudo);
            x = x->dir;
        }
    }
}
```

Varredura

```
void erd_i (no *r) {
    int t=0;
    no *x, *p[100]; //p é uma pilha [0..t-1] elementos
    x = r;

    while (x!=NULL || t>0) {
        if (x!=NULL){
            p[t++] = x;
            x = x->esq;
        }
        else {
            x = p[--t];
            printf("%d\n", x->conteudo);
            x = x->dir;
        }
    }
}
```

Varredura

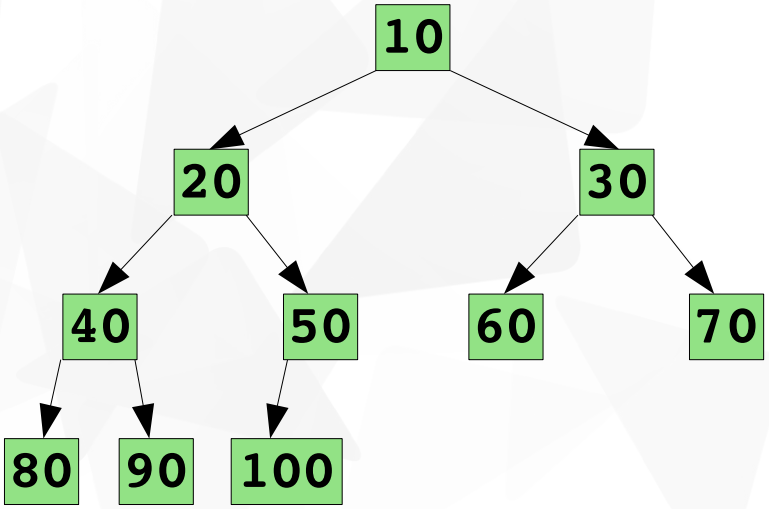
A versão usa uma **pilha** p[0..t-1] de endereços e mais um endereço x que é candidato a entrar na pilha; é como se a pilha fosse

p[0], p[1], . . . , p[t-1], x .

A sequência x, p[t-1], . . . , p[0] é uma espécie de "roteiro" daquilo que ainda precisa ser feito:

- x representa a instrução "imprima a árvore x" e
- cada p[i] representa a instrução "imprima o vértice p[i] e em seguida a árvore p[i]->dir".

Para dimensionar a pilha, suporemos que nossa árvore binária não tem mais que 100 vértices.




```c
void erd_i2 (no *r) {
    int t=0, k;
    no *x, *p[100]; //p é uma pilha [0..t-1] elementos
    x = r;

    while (x!=NULL || t>0) {
        printf("\n");
        for (k=0; k<t; k++)
            printf("%d ", p[k]->conteudo);

        if (x!=NULL){
            p[t++] = x;
            x = x->esq;
        }
        else {
            x = p[--t];
            printf("\t\t imprimir=%d", x->conteudo);
            x = x->dir;
        }
    }
}
```

```
10
10 20
10 20 40
10 20 40 80          imprimir=80
10 20 40             imprimir=40
10 20
10 20 90             imprimir=90
10 20                imprimir=20
10
10 50
10 50 100            imprimir=100
10 50                imprimir=50
10                   imprimir=10

30
30 60                imprimir=60
30                   imprimir=30

70                   imprimir=70
```

Altura de uma árvore

Altura de uma árvore

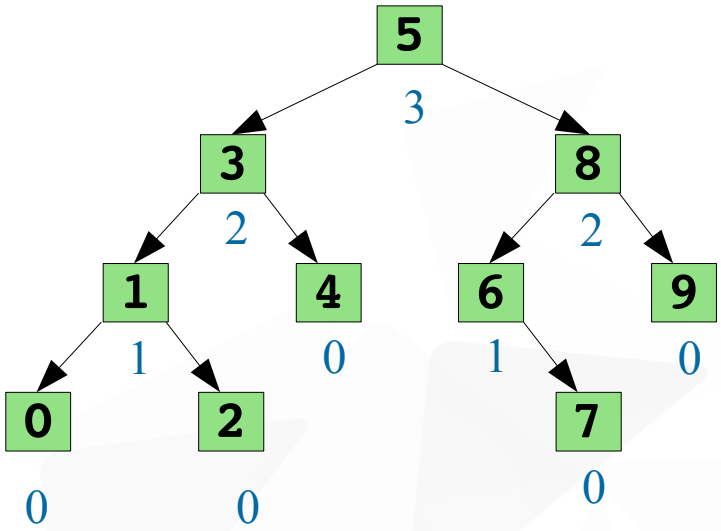
A altura de um nó **x** em uma **árvore binária** é a distância entre x e o seu descendente mais afastado.

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

A altura de uma árvore vazia é -1

Altura de uma árvore

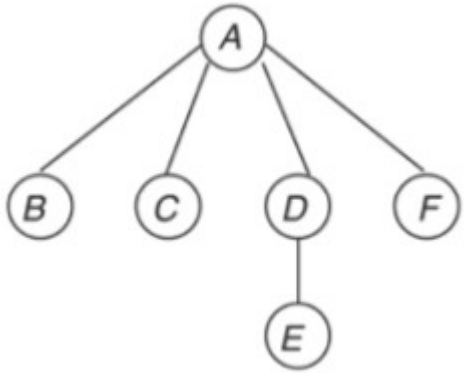
```
int altura (no *r) {
    if (r==NULL)
        return -1;
    else {
        int he = altura(r->esq);
        int hd = altura(r->dir);
        if (he<hd)
            return hd+1;
        else
            return he+1;
    }
}
```



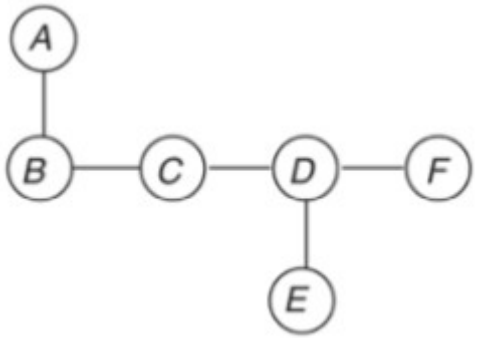
Conversão de uma floresta

Conversão de uma floresta

Árvore n-aria

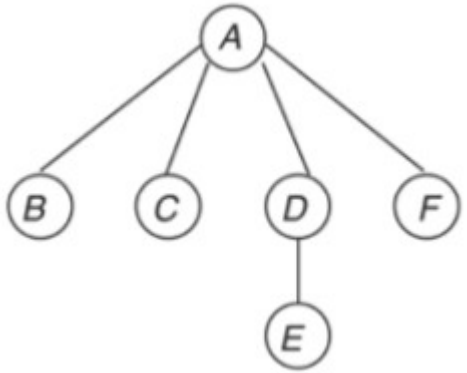


Árvore binária

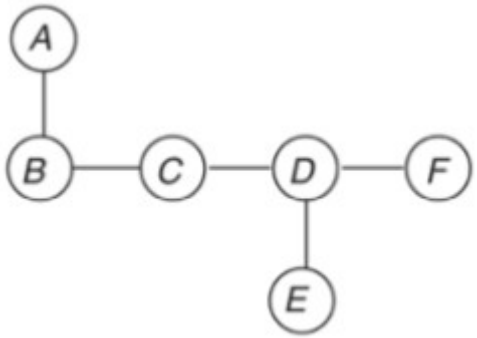


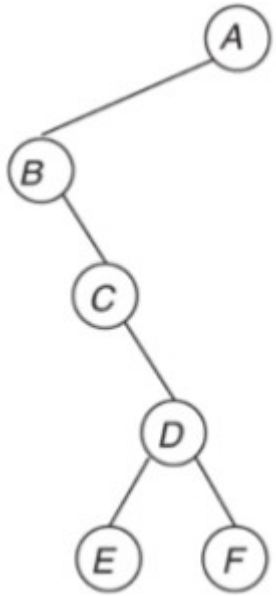
Conversão de uma floresta

Árvore n-aria

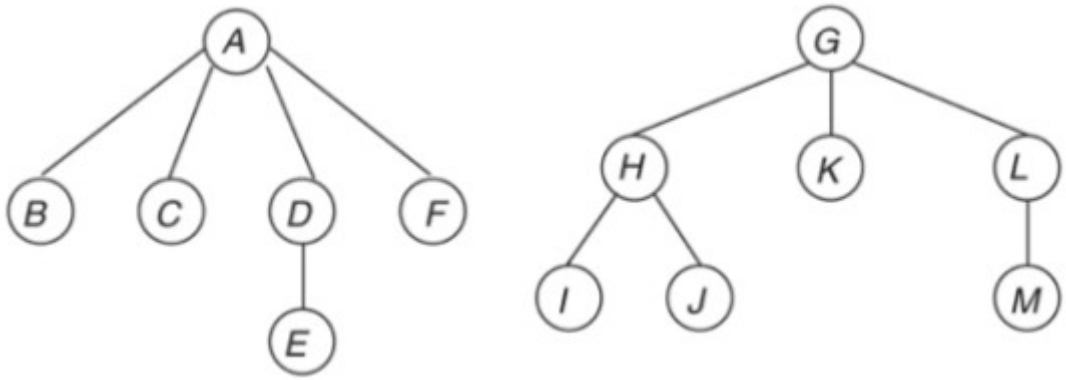


Árvore binária

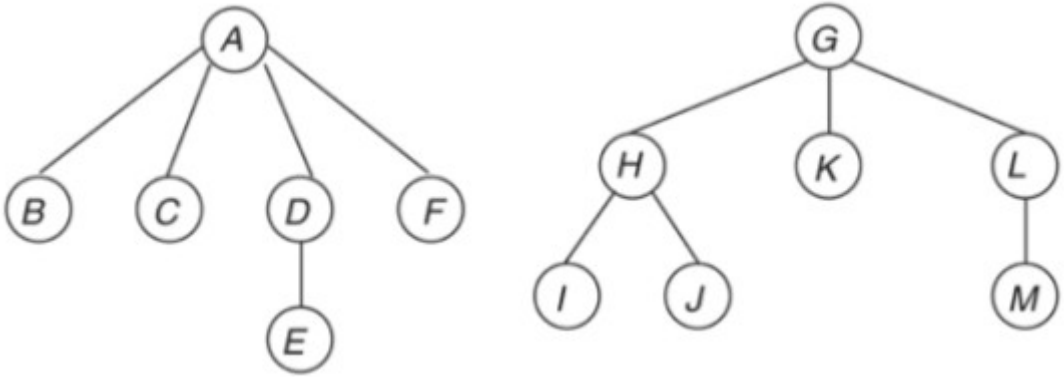




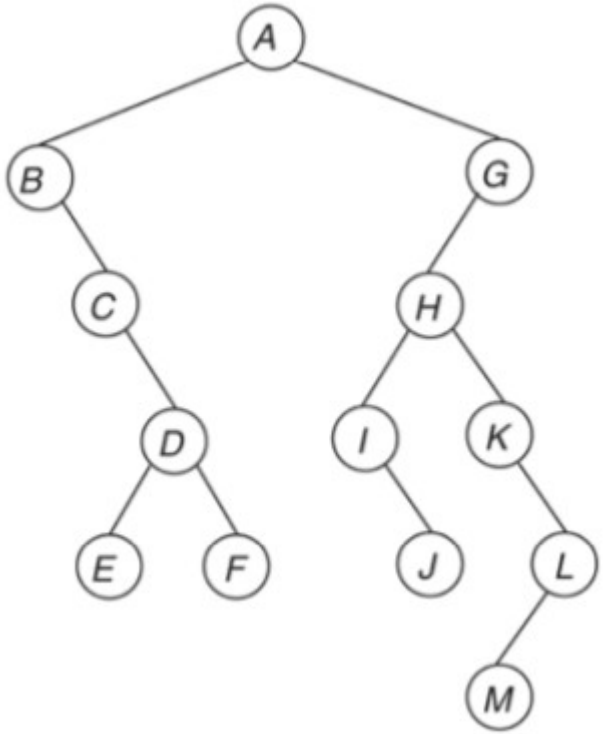
Floresta



Floresta

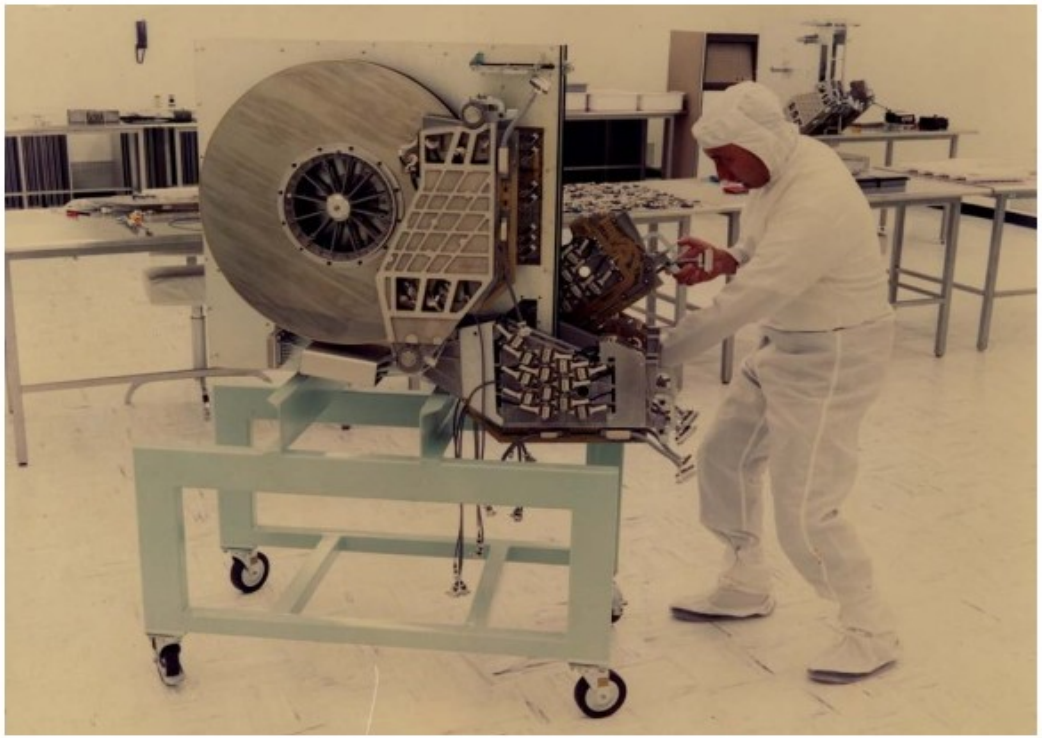


Árvore binária



In September 1956 IBM launched the 305 RAMAC, the first ‘SUPER’ computer with a hard disk drive (HDD). The HDD weighed over a ton and stored 5 MB of data.’

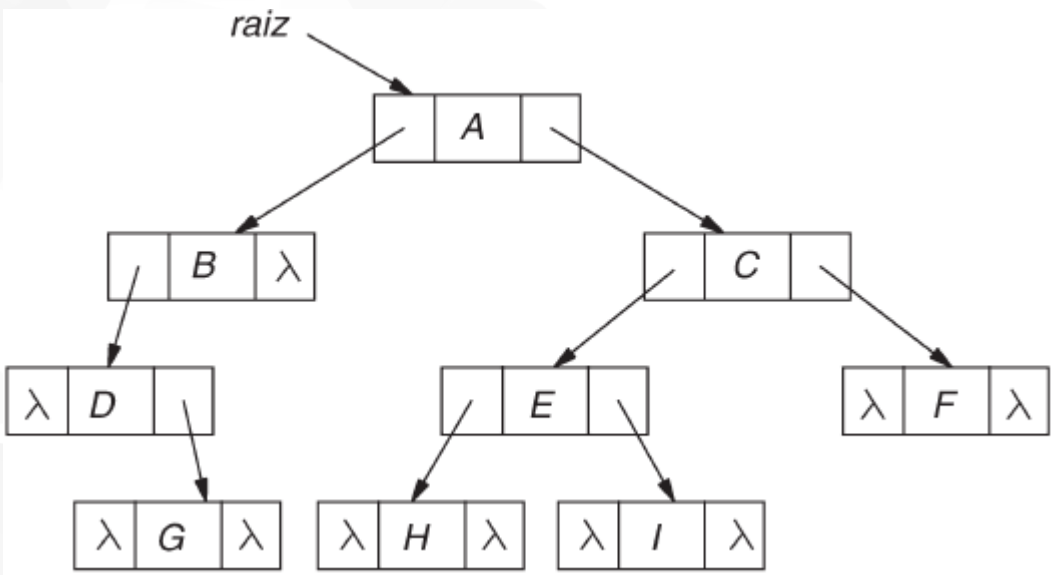
Case in point, here below is a 250 MB hard disk drive from 1979.



Árvores binárias com costura
Árvores binárias com fios
Threaded binary tree

Representação de uma árvore binária

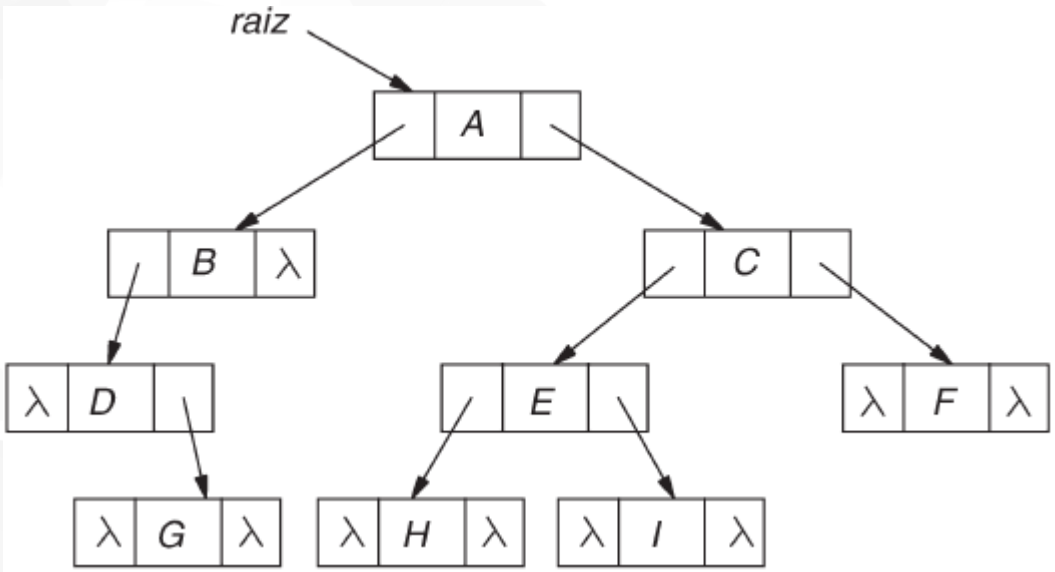
λ =NULL



Para uma árvore binária de n vértices:
São requeridas 2n+1 unidades de memória para sua representação

Representação de uma árvore binária

λ =NULL

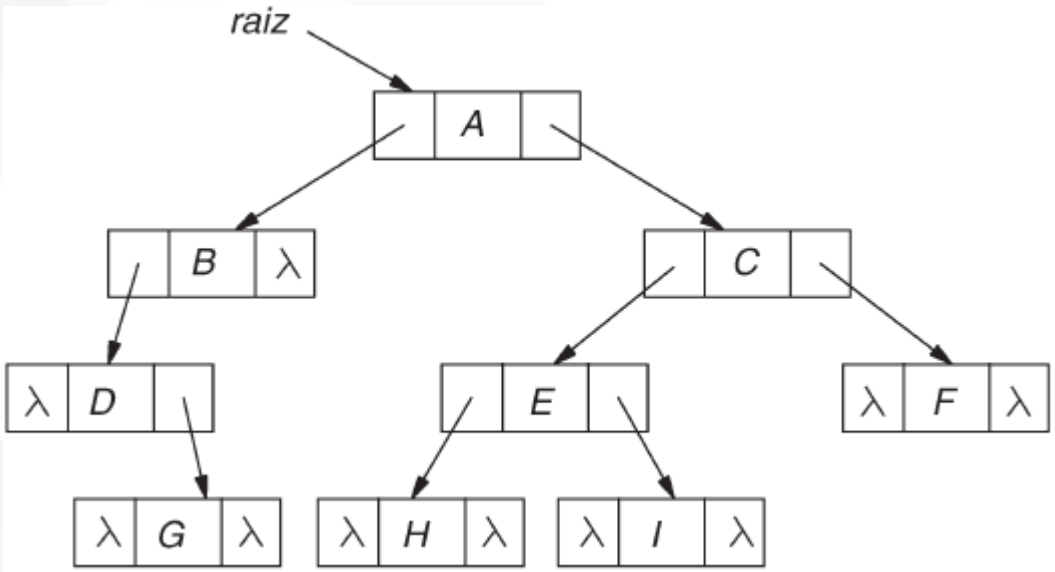


Para uma árvore binária de n vértices:
São requeridas 2n+1 unidades de memória para sua representação

n+1 unidades de memória são iguais a NULL.

Representação de uma árvore binária

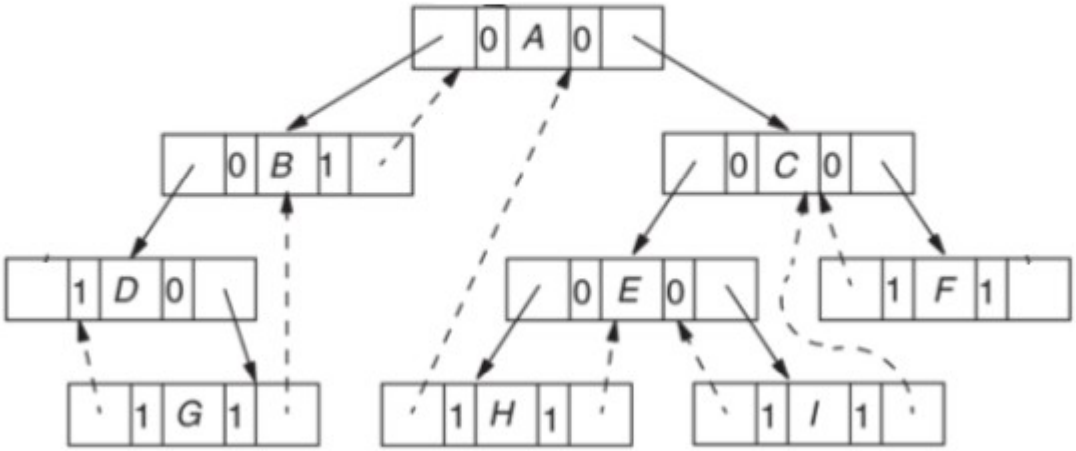
λ =NULL



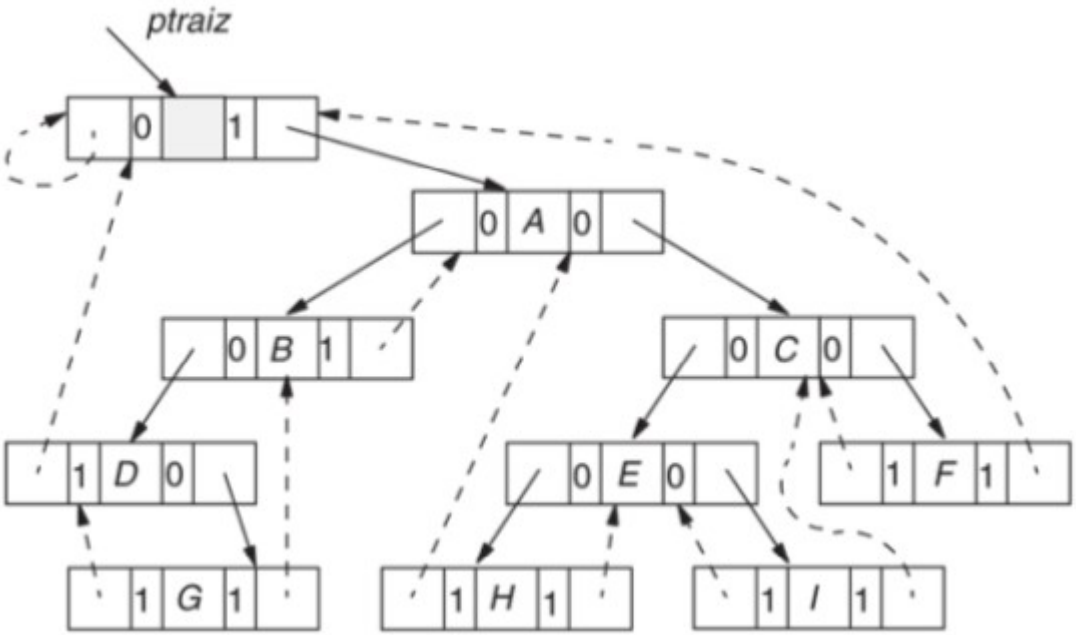
Para uma árvore binária de n vértices:
São requeridas 2n+1 unidades de memória para sua representação

n+1 unidades de memória são iguais a NULL.
Por que não aproveitar esse espaço de memoria?

Árvore binária com costura



Árvore binária com costura



Sem uso de uma pilha é facil percorrer a árvore.

Árvore binária com costura

```
struct cel {
    int conteudo;
    struct cel *esq;
    struct cel *dir;
    short int ecostura;
    short int dcostura;
};
typedef struct cel no;
typedef no *arvore;
```

Desafio 02 – opcional – 0,5 na média final da disciplina
Envio até 25/06 (23h50-Tidia)

- Implemente o algoritmo costurar_erd.

```
// funcao que recebe a raiz r de uma árvore binária
// e retorna sua versão com costura
no * costurar_erd (no *r) {
    //...
}
```

- Veja arquivo desafio2.c no repositório do Tidia (aula 06).
- Apresentação livre de exemplos (quanto mais completo melhor).

- Apenas 2 arquivos que deverá submeter pelo Tidia:
  - Código fonte em C/C++ (RA_desafio2.c/cpp)
  - Um PDF contendo uma simples descrição do programa (não maior a 4 páginas). O formato desse relatório é livre. (RA_desafio2.pdf)