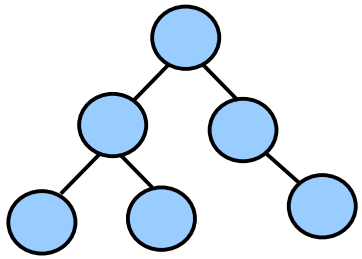
INF01203 – Estruturas de Dados

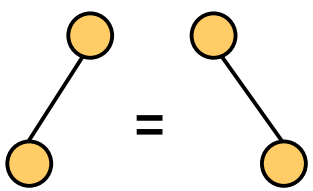
Árvores Binárias

Árvores Binárias

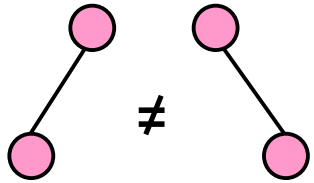
- grau dos nós
  - 0 – 1 – 2



- ordenadas
  - sub-árvore da esquerda ≠ sub-árvore da direita

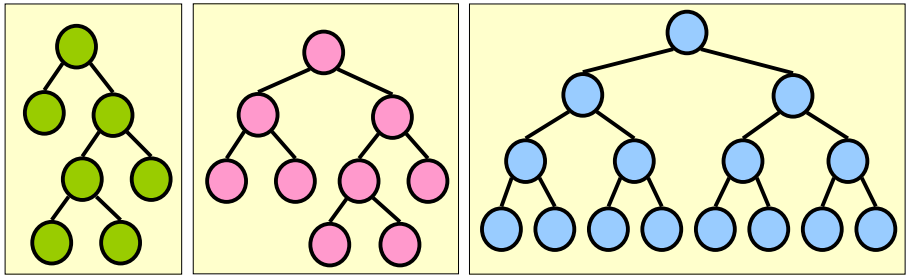


Árvore qualquer



Árvore Binária

Tipos de Árvores Binárias

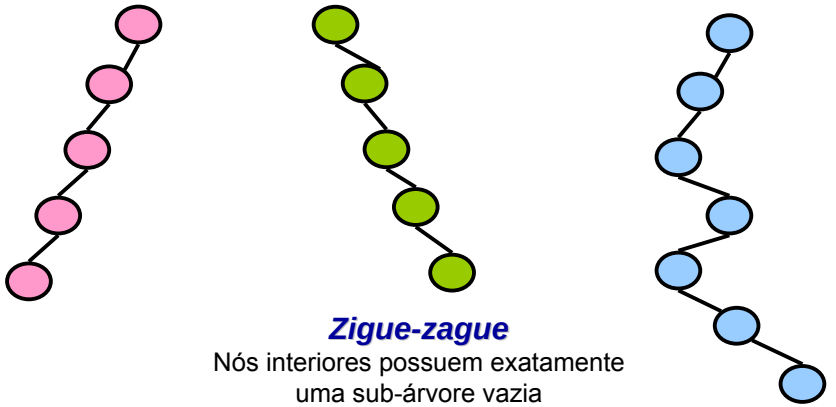


Estritamente Binária
0 ou 2 filhos

Binária Completa
Sub-árvores vazias no último ou penúltimo nível

Binária Cheia
Sub-árvores vazias somente no último nível

Tipos de Árvores Binárias



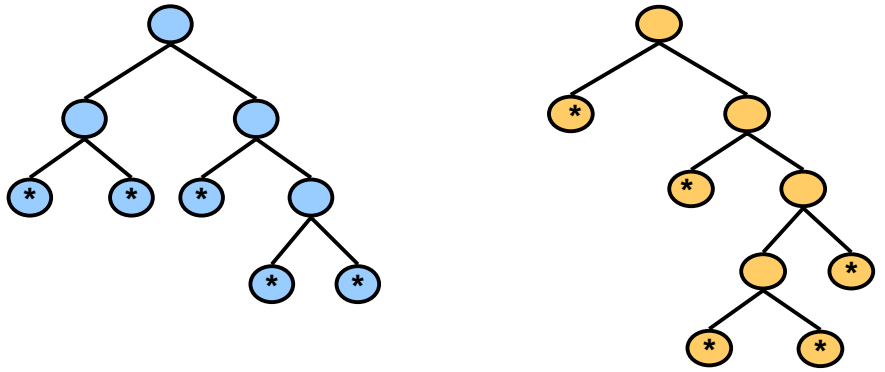
Zigue-zague
Nós interiores possuem exatamente uma sub-árvore vazia

Tipos de Árvores Binárias

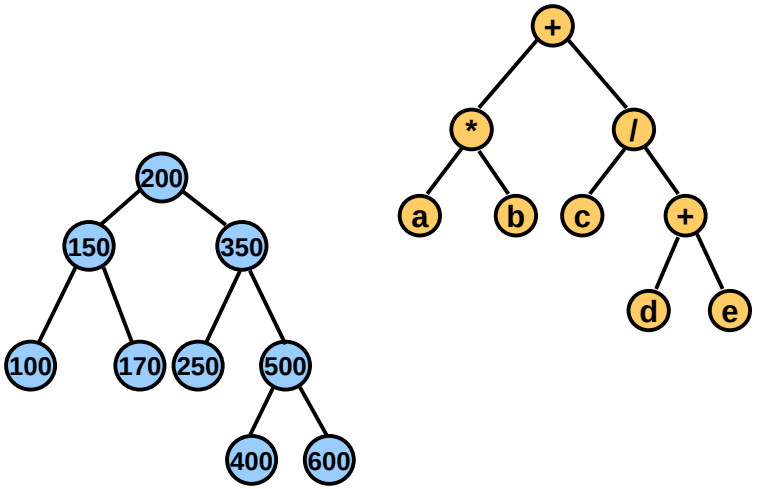
- Dados os tipos de árvores:
  - estritamente binária
  - completa
  - cheia
  - zigue-zague
- Qual árvore possui altura *máxima* ?
  - zigue-zague
- Qual árvore possui altura *mínima* ?
  - cheia

Árvore Binária

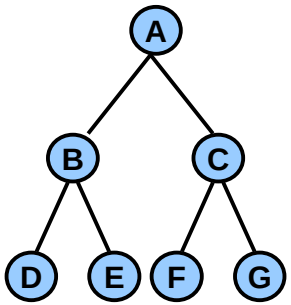
- Comprimento de caminho de uma árvore
  - tempo de execução

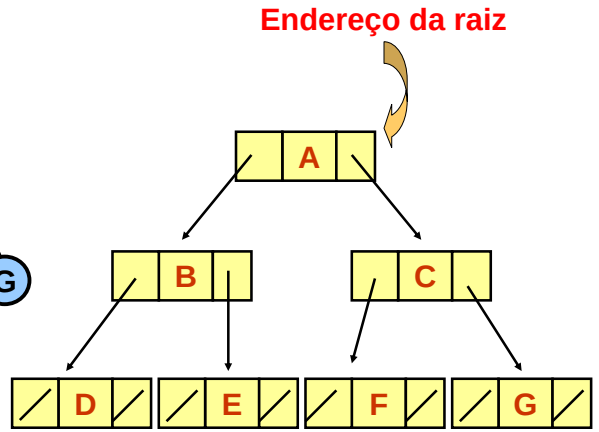


Exemplos de Árvores Binárias



Implementação



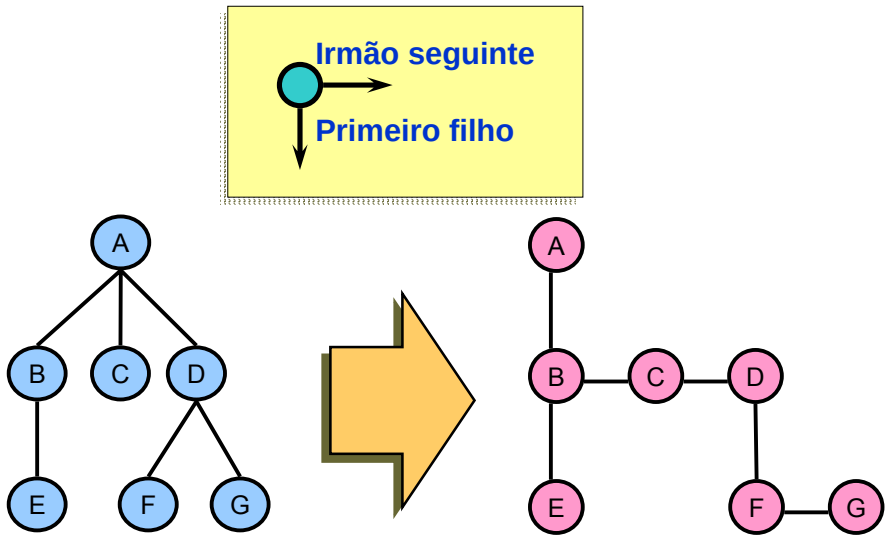


Árvores Binárias - vantagens

- Otimização de alocação de memória
- Mais fáceis de manipular
- Implementação de operações é muito simplificada

Transformar árvore **n-ária** em **binária**

Transformação: n-ária em binária

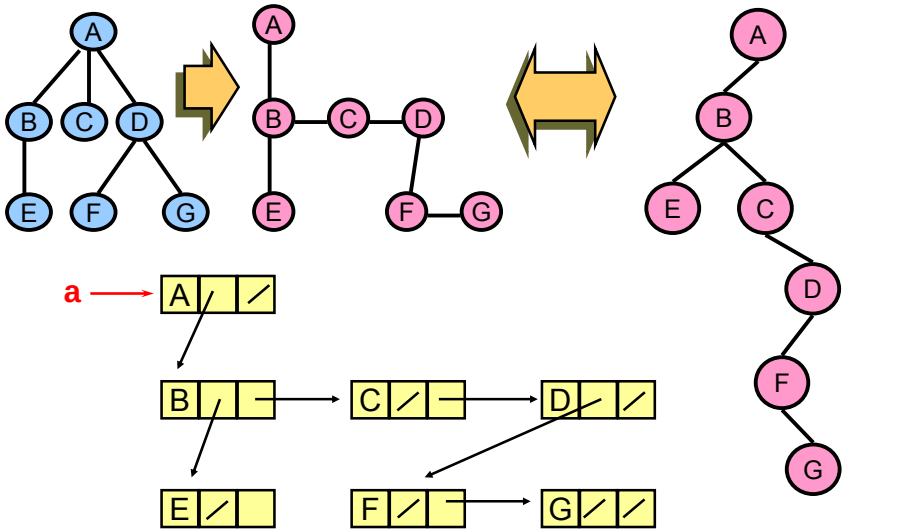


Transformação: n-ária em binária

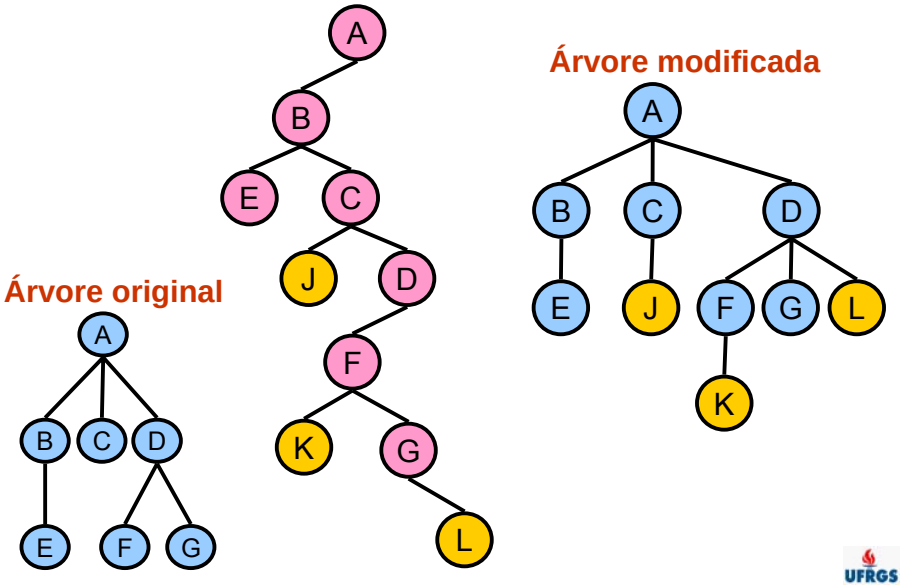
1. O primeiro filho de um nodo passa a ser seu filho à esquerda na árvore binária
2. Os demais filhos de um nodo passam a ser filhos à direita do seu irmão imediato à esquerda
3. Executar o mesmo processo para cada nodo da árvore

Filho à esquerda = primeiro filho
Filho à direita = irmão seguinte

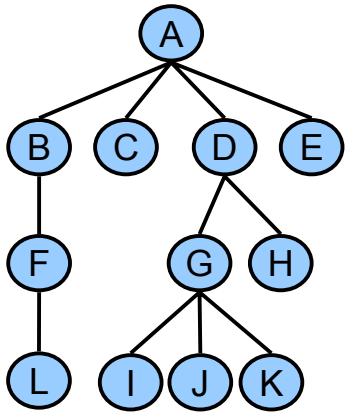
Transformação: n-ária em binária



Reconstituição Árvore n-ária



Exercícios

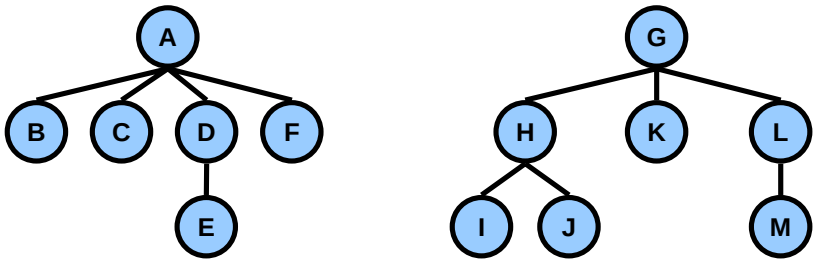


?

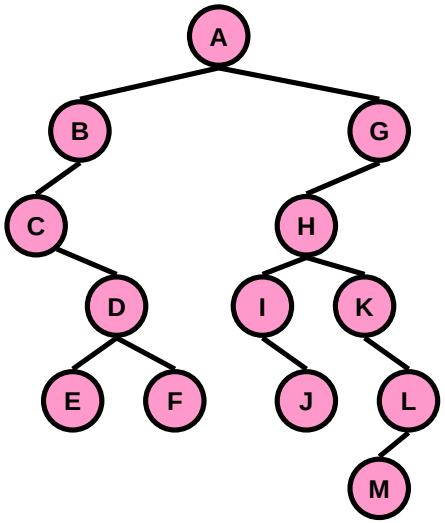
- Converter Binária
- inserir Y filho direita de A
- inserir X filho esquerda Y
- inserir Z filho direita Y

Transformação de Floresta em Binária

- Para converter uma floresta em árvore binária, basta considerar as raízes das árvores como nós irmãos e aplicar a conversão anterior

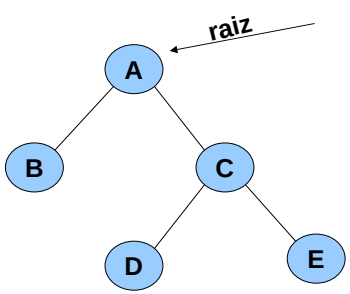


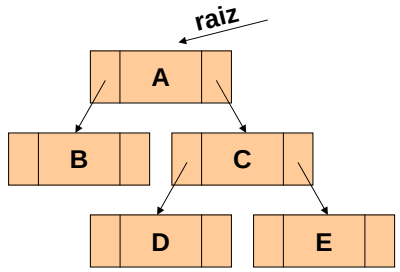
Transformação de Floresta em Binária



Implementação de Árvores Binárias

TAD : Estrutura de Dados





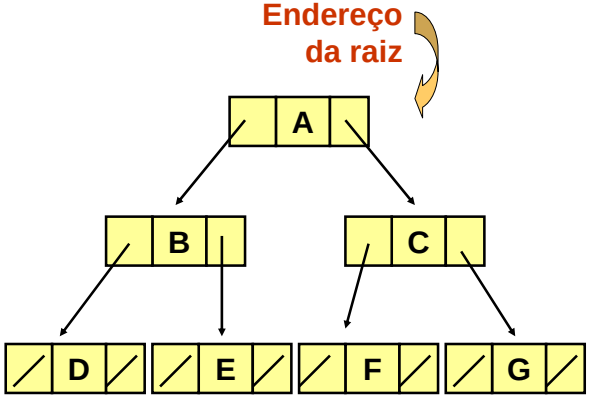
Nodo

Tipo

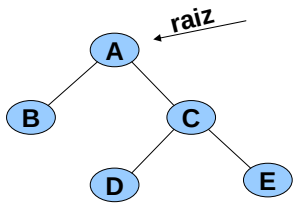
```
Pnodo = ↑Nodo;
Nodo = registro
         esq : Pnodo;
         info: Dados;
         dir : Pnodo
       fim;
```

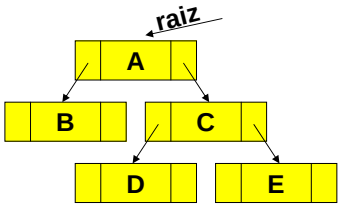

TAD : Operações

- Inicializa
- Insere
  - raiz
  - meio
  - folha
- Consulta
- Remove
- Destrói



Inicializa

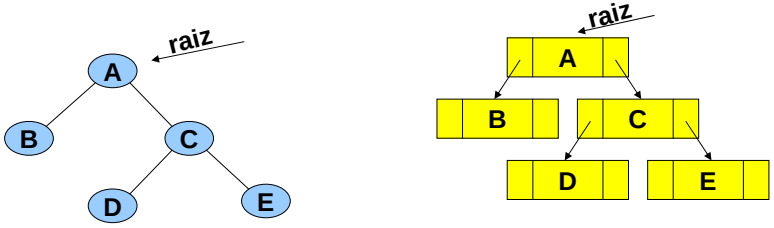




Criar uma árvore vazia

```
proc inicializa(var a: Pnodo);
início
    a := nil
fim inicializa;
```

Insere raiz



Alocar nodo raiz

```
proc insereRaiz(var a: Pnodo; info: Dados);
var raiz: Pnodo;
início
  alocar(raiz);
  raiz↑.esq := raiz↑.dir := nil;
  raiz↑.info := info;
  a := raiz;
fim insereRaiz;
```


Insere: filho à esqueda

```
proc insereEsq(var a: Pnodo; infoPai, infoFilho: Dados);
var pai, novo: Pnodo;
início
  pai := localiza(a, infoPai);
  se pai ≠ nil
     então
       se pai↑.esq ≠ nil
          então erro("Nodo já possui sub-árvore esquerda")
          senão início {insere o novo nodo}
                  aloca(novo);
                  novo↑.dir  := novo↑.esq := nil;
                  novo↑.info := infoFilho;
                  pai↑.esq   := novo;
                fim;
fim insereEsq;
```


Insere: filho à direita

```
proc insereDir(var a: Pnodo; infoPai, infoFilho: Dados);
var pai, novo: Pnodo;
início
  pai := localiza(a, infoPai);
  se pai ≠ nil
     então
       se pai↑.dir ≠ nil
          então erro("Nodo já possui sub-árvore direita")
          senão início {insere o novo nodo}
                  aloca(novo);
                  novo↑.dir  := novo↑.esq := nil;
                  novo↑.info := infoFilho;
                  pai↑.dir   := novo;
                fim;
fim insereDir;
```


TAD: Operações sobre Árvores Binárias

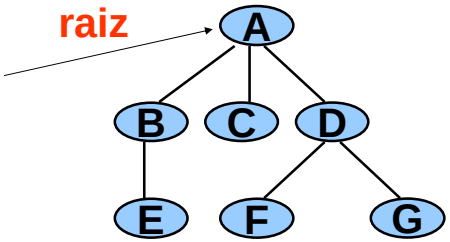
```
Pnodo = ↑Nodo;
Nodo = registro
         esq : Pnodo;
         info: Dados;
         dir : Pnodo
       fim;

proc inicializa(var a: Pnodo);
// cria uma árvore vazia
proc insereRaiz(var a: Pnodo; info: Dados);
// aloca nodo raiz e insere dado
proc insereEsq(var a: Pnodo; InfoPai, InfoFilho: Dados);
// insere um nodo na sub-árvore esquerda
proc insereDir(var a: Pnodo; InfoPai, InfoFilho: Dados);
// insere um nodo na sub-árvore direita
```

Caminhamentos em Árvores Binárias

Consulta Nodos

- acesso sempre através da raiz
- cada nodo deve ser "visitado" uma vez, e apenas uma vez

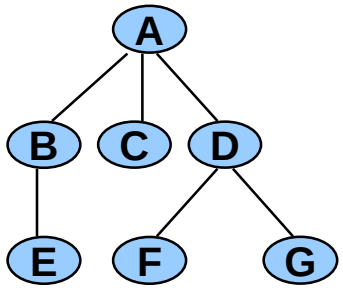


Visita **a um nodo**

acesso a um nodo
para realizar alguma operação

Caminhamentos

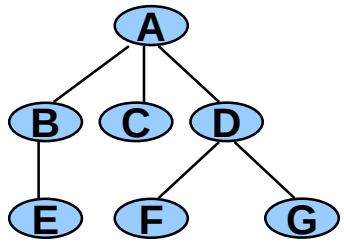
- método de percurso sistemático de todos os nodos de uma árvore, de modo a que cada nodo seja visitado exatamente uma vez



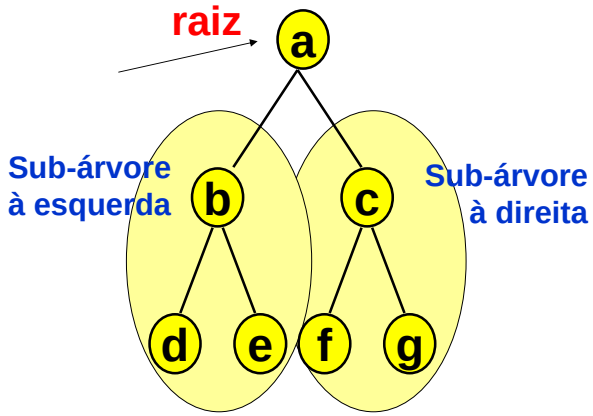
Caminhamentos

- um caminhamento define uma seqüência de nodos
- cada nodo passa a ter um nodo seguinte, ou um nodo anterior, ou ambos (exceto árvore com 1 só nodo)
- seqüência de nodos depende do caminhamento

Ex: Caminhamento 1:
A - B - C - D - E - F - G
Caminhamento 2:
A - B - E - C - D - F - G



Principais Caminhamentos



Caminhamentos

Pré-Fixado à esquerda
- .Visita a raiz
- .Percorre a sub-árvore esquerda
- .Percorre a sub-árvore direita

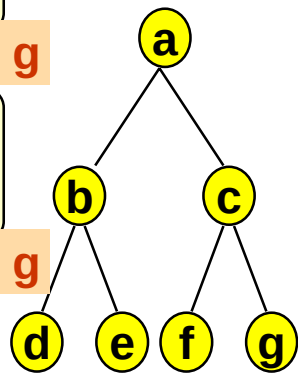
a - b - d - e - c - f - g

Central à esquerda
- .Percorre a sub-árvore esquerda
- .Visita a raiz
- .Percorre a sub-árvore direita

d - b - e - a - f - c - g

Pós-Fixado à esquerda
- .Percorre a sub-árvore esquerda
- .Percorre a sub-árvore direita
- .Visita a raiz

d - e - b - f - g - c - a



Caminhamentos

Pré-Fixado à direita
- .Visita a raiz
- .Percorre a sub-árvore direita
- .Percorre a sub-árvore esquerda

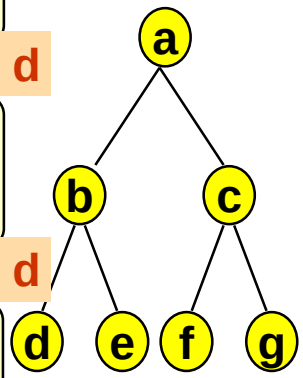
a - c - g - f - b - e - d

Central à direita
- .Percorre a sub-árvore direita
- .Visita a raiz
- .Percorre a sub-árvore esquerda

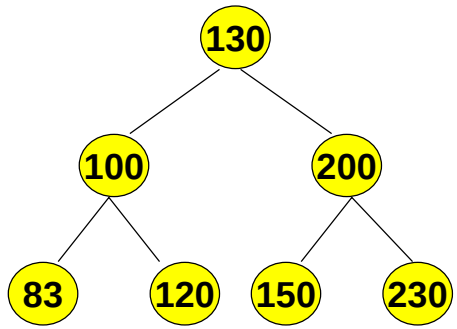
g - c - f - a - e - b - d

Pós-Fixado à direita
- .Percorre a sub-árvore direita
- .Percorre a sub-árvore esquerda
- .Visita a raiz

g - f - c - e - d - b - a



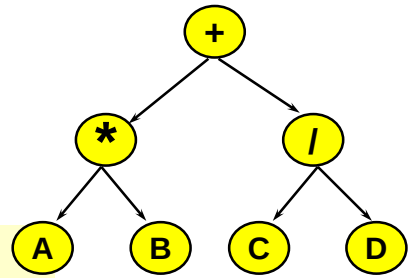
Exemplo 02



.Central à esquerda?

.Central à direita?

Exemplo: expressão aritmética



Pré-fixado E
- .raiz
- .esquerda
- .direita

Central E
- .esquerda
- .raiz
- .direita

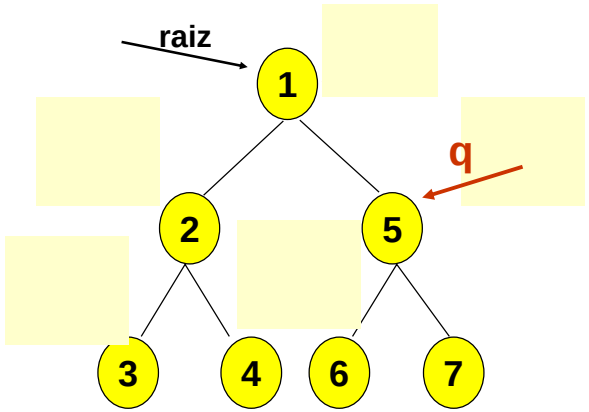
Pós-Fixado E
- .esquerda
- .direita
- .raiz

A * B + C / D

\+ * A B / C D

A B * C D / +

Percorrer: Pré-Fixado Esquerda

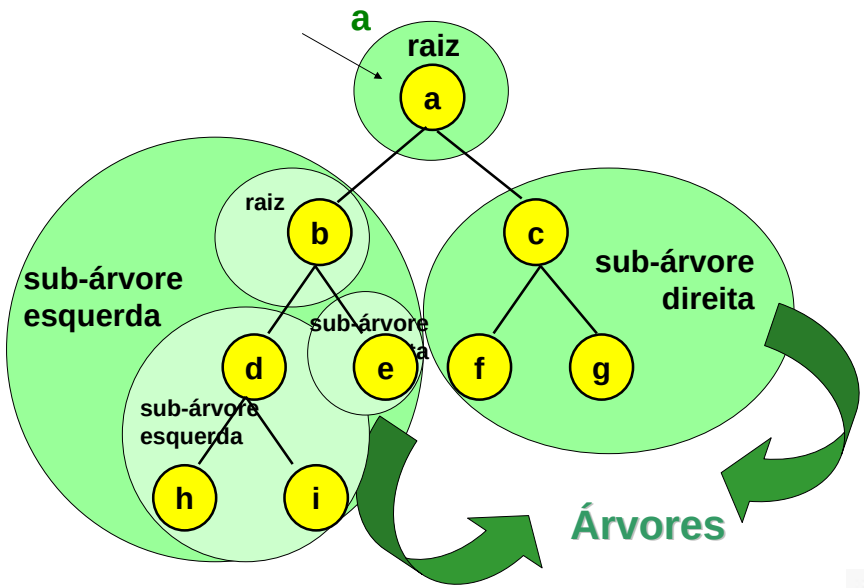


1- 2 - 3 - 4 - 5 ...

Percorrer: Pré-Fixado Esquerda

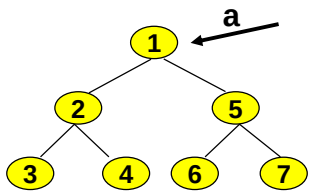
```
proc preFixado(a: Pnodo);
var paux: Pnodo;  {apontador auxiliar}
    s: Pilha;     {simbolizando a pilha}
início
  push(s,a);      {coloca na pilha o endereço da raiz}
  enquanto (consulta(s,paux)=true)
  {enquanto houver alguma coisa na pilha}
  faça início
    se(pop(s,paux)=true) {tira endereço do topo da pilha}
    então início
      visita(paux);       {efetua a operação desejada no nodo}
      push(s,paux↑.dir); {empilha raiz da sub-árvore direita}
      push(s,paux↑.esq)  {empilha raiz da sub-árvore esquerda}
    fim
  fim
fim preFixado;
```


Recursividade em Árvores



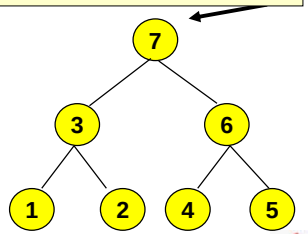
Pré-Fixado Esquerda

```
proc preFixado(a: pNodo);
{percurso pré-fixado esquerdo, usando recursividade}
início
  se a <> nil          {existe a árvore ou sub-árvore}
  então início
    visita(a);         {executa operação}
    preFixado(a↑.esq); {percurso da sub-árvore esquerda}
    preFixado(a↑.dir)  {percurso da sub-árvore direita}
  fim
fim preFixado;
```



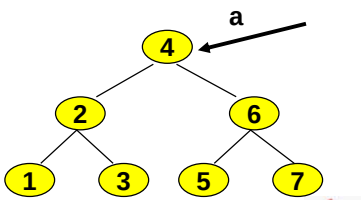
Pós-Fixado a Esquerda

```
proc posFixado(a: pNodo);
{percurso pos-fixado esquerdo, usando recursividade}
início
  se a <> nil          {existe a árvore ou sub-árvore}
  então início
    posFixado(a↑.esq); {percurso da sub-árvore esquerda}
    posFixado(a↑.dir)  {percurso da sub-árvore direita}
    visita(a);         {executa operação}
  fim
fim posFixado;
```



Central Esquerda

```
proc central(a: pNodo);
{percurso central à esquerda, usando recursividade}
início
  se a <> nil        {existe a árvore ou sub-árvore}
  então início
    central(a↑.esq); {percurso da sub-árvore esquerda}
    visita(a);       {executa operação}
    central(a↑.dir)  {percurso da sub-árvore direita}
  fim
fim central;
```



TAD: Operações sobre Árvores Binárias (parcial)

```
proc inicializa(var a: Pnodo);
// cria uma árvore vazia
proc insereRaiz(var a: Pnodo; info: Dados);
// aloca nodo raiz e insere dado
proc insereEsq(var a: Pnodo; InfoPai, InfoFilho: Dados);
// insere um nodo na sub-árvore esquerda
proc insereDir(var a: Pnodo; InfoPai, InfoFilho: Dados);
// insere um nodo na sub-árvore direita
proc preFixado(a: pNodo);
// percurso pré-fixado esquerdo, usando recursividade
proc posFixado(a: pNodo);
// percurso pós-fixado esquerdo, usando recursividade
proc central(a: pNodo);
// percurso central à esquerda, usando recursividade
proc constroi(var a: Pnodo);
// constrói uma (sub)árvore e devolve o endereço da raiz
função copia(a: Pnodo):Pnodo;
// monta uma cópia de uma árvore a, devolvendo o endereço da raiz
```