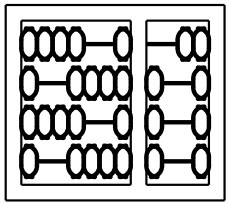


Universidade Estadual de Campinas - UNICAMP
Instituto de Computação - IC

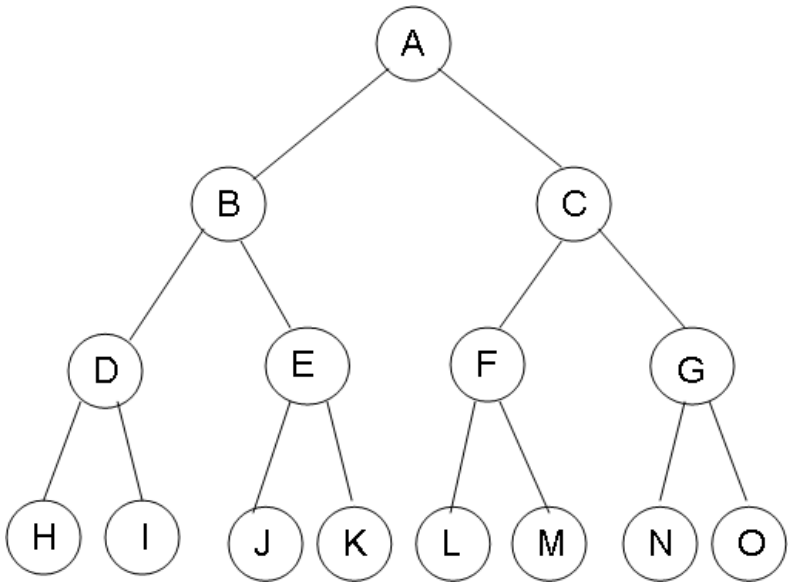
Árvores: Parte 2

Tópicos

1. Árvore binária completa
2. Percursos em árvores binárias
3. Representação e reconstrução de árvores

Árvore completa

• Definição: Dizemos que uma árvore binária de altura h é completa se ela contém $2^h - 1$ nós (número máximo possível como visto anteriormente).



Representação sequencial de uma árvore completa

– Podemos associar a cada nó x da árvore completa um índice $f(x)$ tal que:

Representação sequencial de uma árvore completa

– Podemos associar a cada nó x da árvore completa um índice $f(x)$ tal que:

1. Se x é a raiz, então $f(x) = 0$.

Representação sequencial de uma árvore completa

– Podemos associar a cada nó x da árvore completa um índice $f(x)$ tal que:

1. Se x é a raiz, então $f(x) = 0$.
2. Se x_e e x_d são, respectivamente, os filhos esquerdo e direito de x, então $f(x_e) = 2f(x) + 1$ e $f(x_d) = 2f(x) + 2$.

Representação sequencial de uma árvore completa

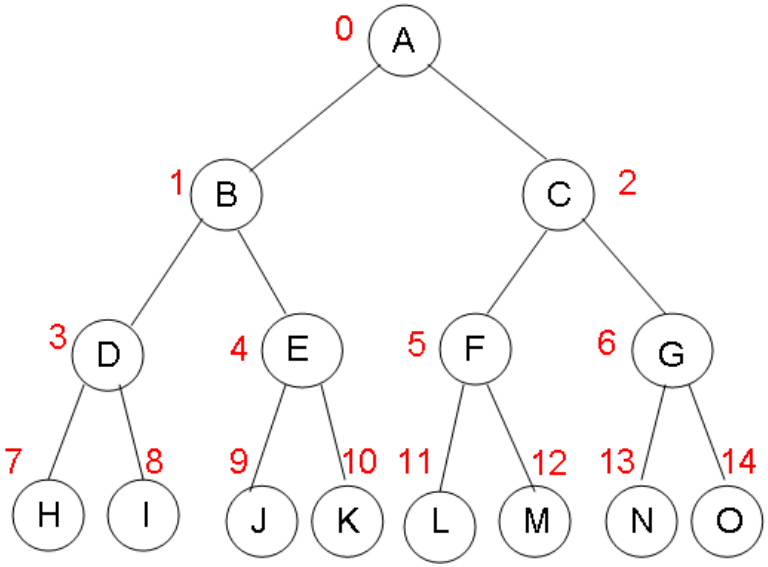
– Podemos associar a cada nó x da árvore completa um índice $f(x)$ tal que:

1. Se x é a raiz, então $f(x) = 0$.
2. Se x_e e x_d são, respectivamente, os filhos esquerdo e direito de x, então $f(x_e) = 2f(x) + 1$ e $f(x_d) = 2f(x) + 2$.
3. Se x não é raiz e x_p é o pai de x, então $f(x_p) = \lfloor \frac{f(x)-1}{2} \rfloor$.

Representação sequencial de uma árvore completa

– Podemos associar a cada nó x da árvore completa um índice $f(x)$ tal que:

1. Se x é a raiz, então $f(x) = 0$.
2. Se x_e e x_d são, respectivamente, os filhos esquerdo e direito de x, então $f(x_e) = 2f(x) + 1$ e $f(x_d) = 2f(x) + 2$.
3. Se x não é raiz e x_p é o pai de x, então $f(x_p) = \lfloor \frac{f(x)-1}{2} \rfloor$.



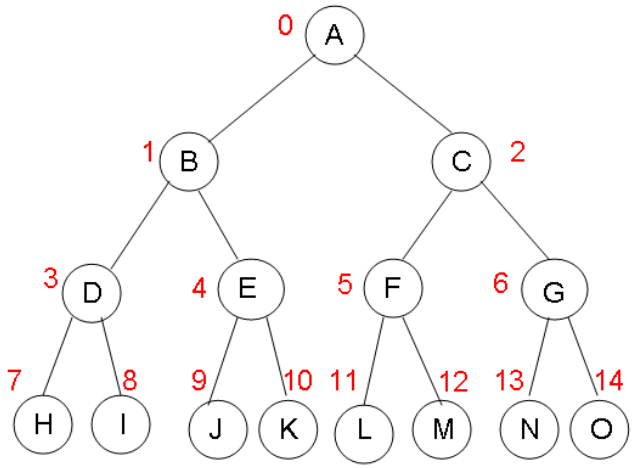
Representação sequencial de uma árvore completa

- Assim, pode-se representar a árvore completa como um vetor de nós:

Representação sequencial de uma árvore completa

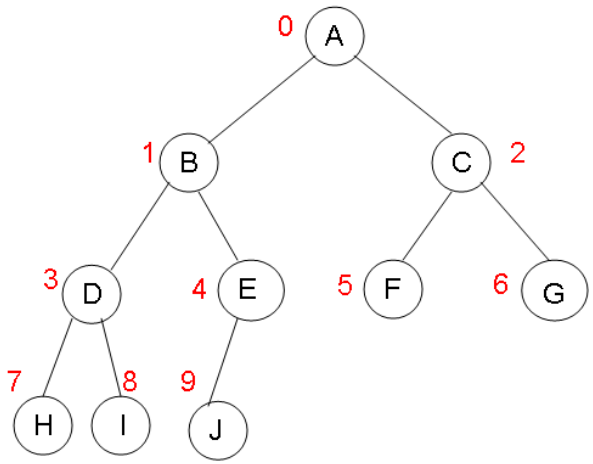
• Assim, pode-se representar a árvore completa como um vetor de nós:

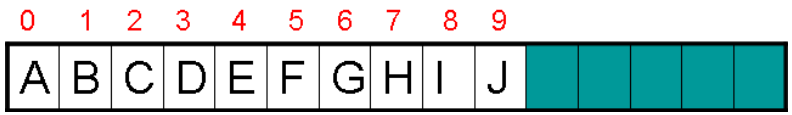
– Vantagens: elimina apontadores $\longrightarrow$ economia de espaço no armazenamento da árvore.



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	B	C	D	E	F	G	H	I	J	K	L	M	N	O

• A representação anterior pode ser estendida a árvores quase completas em que falta um certo número de nós no final da ordem indicada pelo índice $f(x)$.





Percursos em árvores binárias

• Visita sistemática das informações contidas em todos os nós da árvore tal que cada nó recebe uma ou mais visitas.

– Percurso em profundidade: segue a definição recursiva da árvore:

1. pré-ordem:
   - visita a raiz
   - percorre a subárvore esquerda em pré-ordem
   - percorre a subárvore direita em pré-ordem
2. pós-ordem:
   - percorre a subárvore esquerda em pós-ordem
   - percorre a subárvore direita em pós-ordem
   - visita a raiz
3. inordem (ordem simétrica):
   - percorre a subárvore esquerda em inordem
   - visita a raiz
   - percorre a subárvore direita em inordem

Percursos em árvores binárias

• Visita sistemática das informações contidas em todos os nós da árvore tal que cada nó recebe uma ou mais visitas.

– Percurso em profundidade: segue a definição recursiva da árvore:

1. pré-ordem:
   - visita a raiz
   - percorre a subárvore esquerda em pré-ordem
   - percorre a subárvore direita em pré-ordem
2. pós-ordem:
   - percorre a subárvore esquerda em pós-ordem
   - percorre a subárvore direita em pós-ordem
   - visita a raiz
3. inordem (ordem simétrica):
   - percorre a subárvore esquerda em inordem
   - visita a raiz
   - percorre a subárvore direita em inordem

Percursos em árvores binárias

• Visita sistemática das informações contidas em todos os nós da árvore tal que cada nó recebe uma ou mais visitas.

– Percurso em profundidade: segue a definição recursiva da árvore:

1. pré-ordem:
   - visita a raiz
   - percorre a subárvore esquerda em pré-ordem
   - percorre a subárvore direita em pré-ordem
2. pós-ordem:
   - percorre a subárvore esquerda em pós-ordem
   - percorre a subárvore direita em pós-ordem
   - visita a raiz
3. inordem (ordem simétrica):
   - percorre a subárvore esquerda em inordem
   - visita a raiz
   - percorre a subárvore direita em inordem

Percursos em árvores binárias

• Visita sistemática das informações contidas em todos os nós da árvore tal que cada nó recebe uma ou mais visitas.

– Percurso em profundidade: segue a definição recursiva da árvore:

1. pré-ordem:
   - visita a raiz
   - percorre a subárvore esquerda em pré-ordem
   - percorre a subárvore direita em pré-ordem
2. pós-ordem:
   - percorre a subárvore esquerda em pós-ordem
   - percorre a subárvore direita em pós-ordem
   - visita a raiz
3. inordem (ordem simétrica):
   - percorre a subárvore esquerda em inordem
   - visita a raiz
   - percorre a subárvore direita em inordem

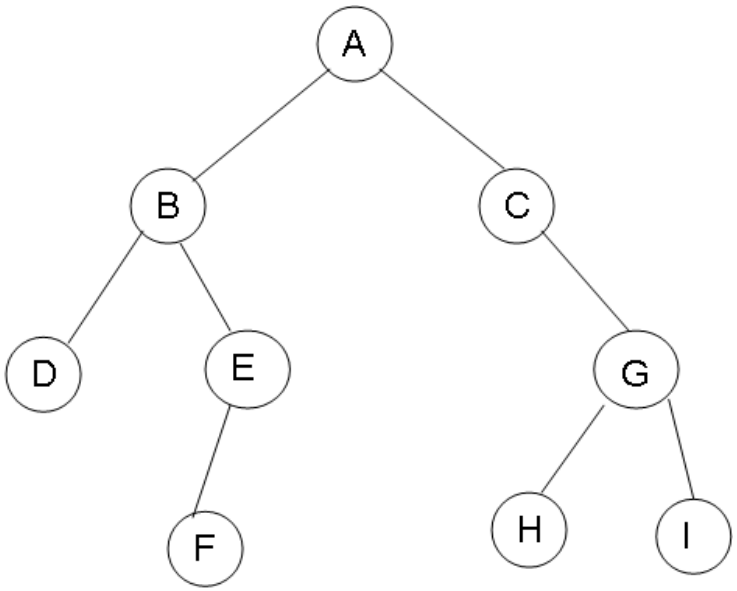
Percursos em árvores binárias

• Visita sistemática das informações contidas em todos os nós da árvore tal que cada nó recebe uma ou mais visitas.

– Percurso em profundidade: segue a definição recursiva da árvore:

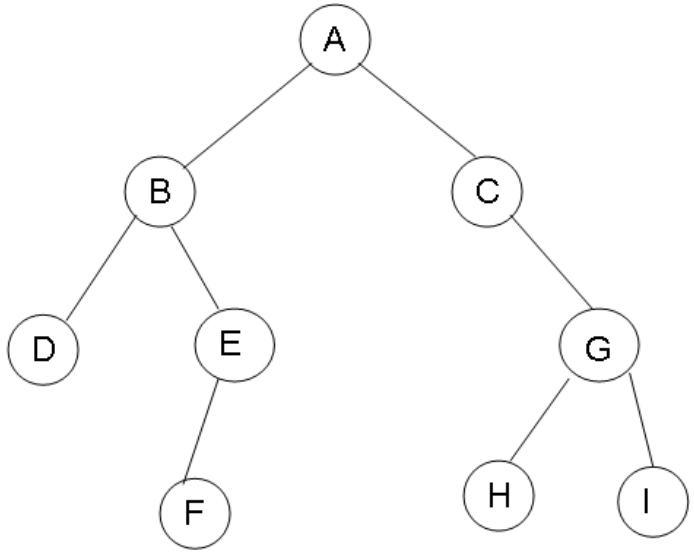
1. pré-ordem:
   - visita a raiz
   - percorre a subárvore esquerda em pré-ordem
   - percorre a subárvore direita em pré-ordem
2. pós-ordem:
   - percorre a subárvore esquerda em pós-ordem
   - percorre a subárvore direita em pós-ordem
   - visita a raiz
3. inordem (ordem simétrica):
   - percorre a subárvore esquerda em inordem
   - visita a raiz
   - percorre a subárvore direita em inordem

- Exemplo:



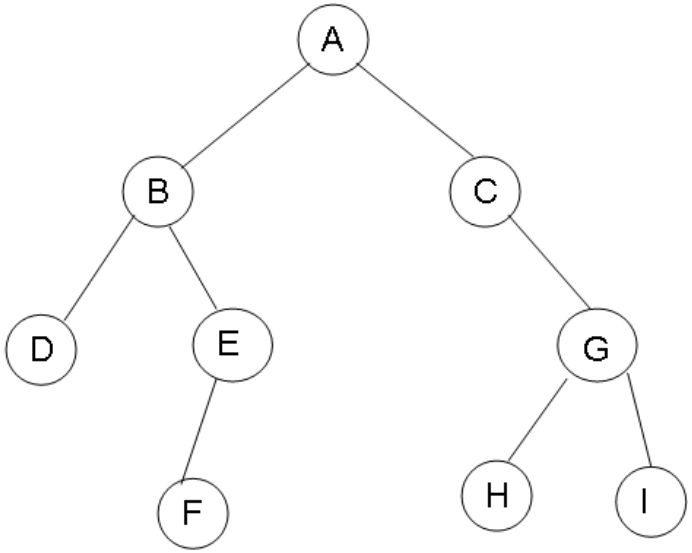
pré-ordem: A B D E F C G H I

- Exemplo:



pós-ordem: D F E B H I G C A

- Exemplo:



inordem: D B F E A C H G I

• Os percursos anteriores podem ser diretamente traduzidos em funções recursivas:

```
/* recebe uma arvore binaria p e imprime o
conteudo de seus nos em pre-ordem */

void PreOrdem (ArvBin *p) {
 if (p != NULL) {
   printf (''%d''\n'', p->info) ;
   PreOrdem (p->esq) ;
   PreOrdem (p->dir) ;
 }
}
```

```
/* recebe uma arvore binaria p e imprime o
conteudo de seus nos em pos-ordem */

void PosOrdem (ArvBin *p) {
 if (p != NULL) {
   PosOrdem (p->esq) ;
   PosOrdem (p->dir) ;
   printf (''%d''\n'', p->info) ;
 }
}
```

```
/* recebe uma arvore binaria p e imprime o
conteudo de seus nos em inordem */

void InOrdem (ArvBin *p) {
 if (p != NULL) {
    InOrdem (p->esq) ;
    printf (''%d''\n'', p->info) ;
    InOrdem (p->dir) ;
 }
}
```

Aplicação

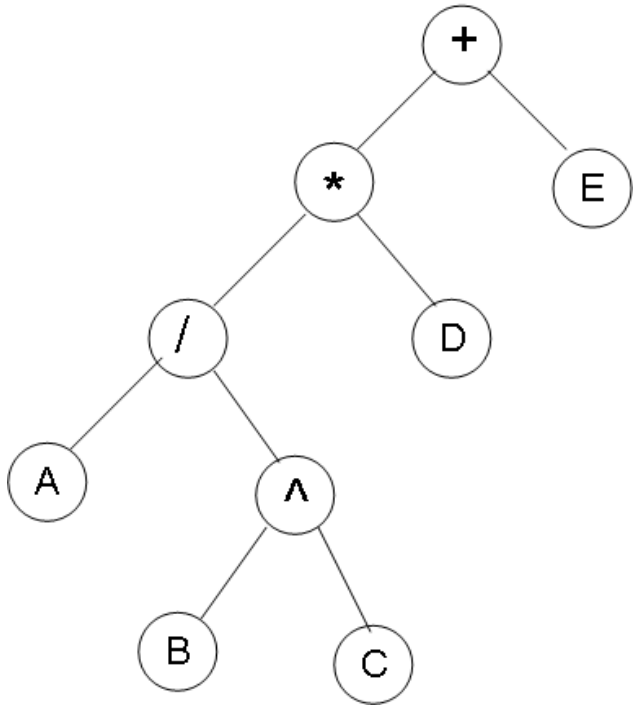
- Representação de expressões aritméticas:

Uma árvore binária pode ser utilizada na representação de expressões aritméticas da seguinte forma: um nó da árvore, associado a um operador, tem duas subárvores, uma para cada operando, e cada folha representa um operando.

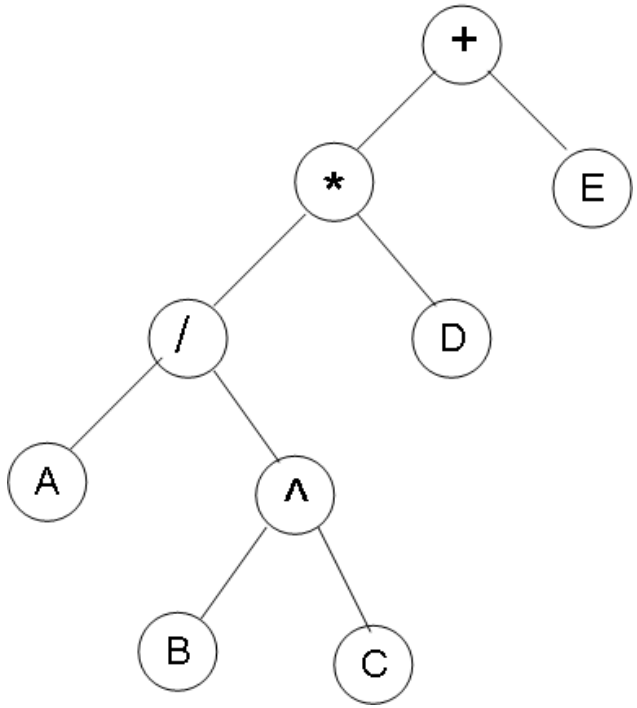
Aplicação

- Representação de expressões aritméticas:

Uma árvore binária pode ser utilizada na representação de expressões aritméticas da seguinte forma: um nó da árvore, associado a um operador, tem duas subárvores, uma para cada operando, e cada folha representa um operando.

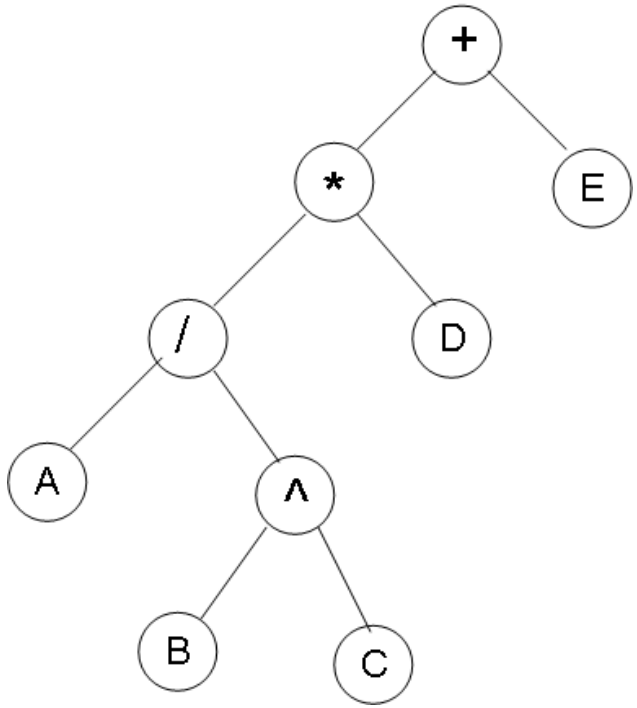


- Avaliação da árvore em *inordem*:



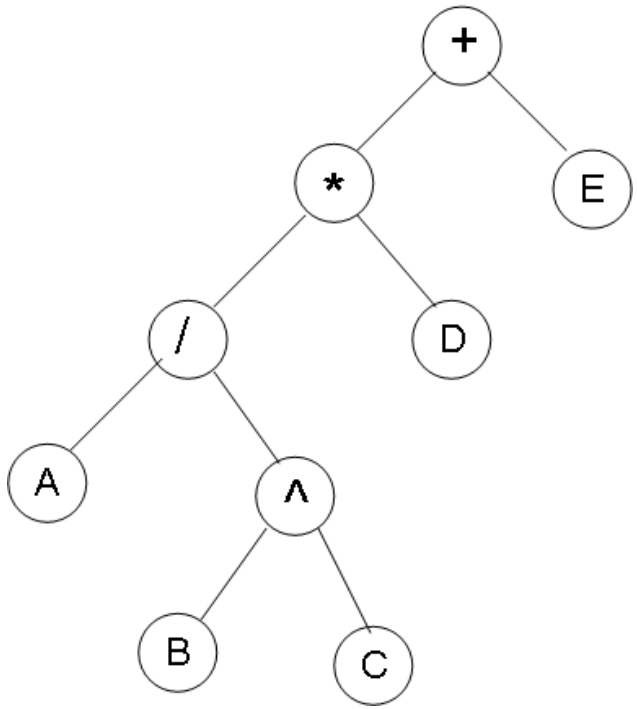

```
inordem: A / B ^ C * D + E   --> representação infixa
```

- Avaliação da árvore em *inordem*:



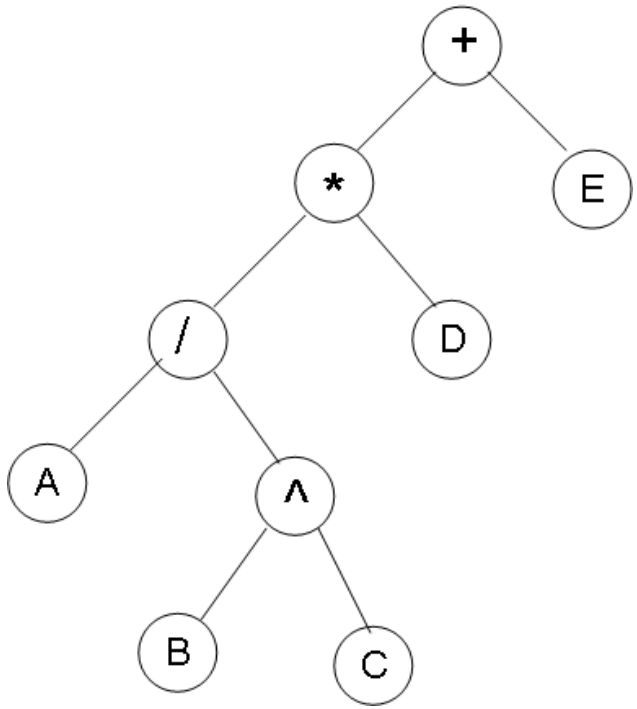

```
inordem: A / B ^ C * D + E   --> representação infixa
```

- Avaliação da árvore em *pré-ordem*:



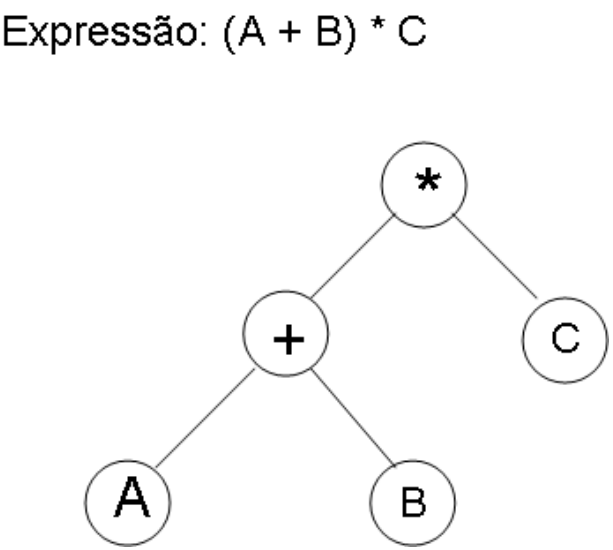

```
pré-ordem: + * / A ^ B C D E   --> representação prefixa
```

- Avaliação da árvore em *pós-ordem*:



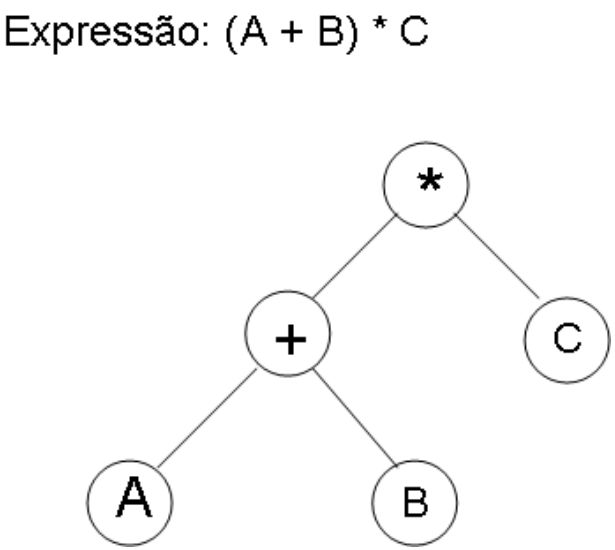

```
pós-ordem: A B C ^ / D * E +   --> representação posfixa
```

- Exemplo 2:



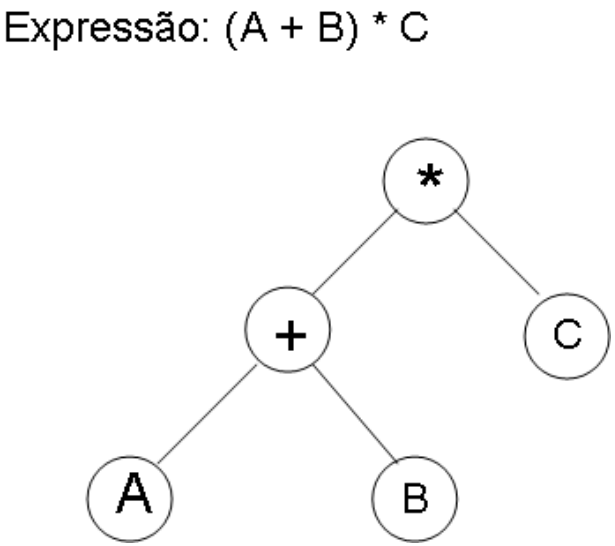

```
pré-ordem: * + A B C
```

- Exemplo 2:




```
pós-ordem: A B + C *
```

- Exemplo 2:



inordem: A + B * C !!!

- Percurso em largura:

  1. Os nós são visitados em ordem de seus níveis, isto é, *nível 1*, depois *nível 2*, e assim por diante.

- Percurso em largura:

  1. Os nós são visitados em ordem de seus níveis, isto é, *nível 1*, depois *nível 2*, e assim por diante.

  2. Dentro de cada nível a ordem é da esquerda para a direita.

- Percurso em largura:

1. Os nós são visitados em ordem de seus níveis, isto é, *nível 1*, depois *nível 2*, e assim por diante.

2. Dentro de cada nível a ordem é da esquerda para a direita.

3. Para implementar, podemos utilizar uma fila que lembra, em certa ordem, os nós que ainda não foram processados.

```
/* Percorre uma árvore binária em largura */
void PercEmLargura (ArvBin *p) {
 Fila f ;
 NoArvBin *q ;
 if (p != NULL) {
   InicializaFila (f) ;
   InsereFila (f, p) ;
   do {
     retira (f, q) ;
     printf (''%d\n'', q->info) ;  /* visita o no */
     if (q->esq != NULL) InsereFila(f, q->esq) ;
     if (q->dir != NULL) InsereFila(f, q->dir) ;
   } while (!FilaVazia(f)) ;
 }
}
```

– O tempo de execução é proporcional ao número de nós $n \rightarrow O(n)$.

```
/* Percorre uma árvore binária em largura */
void PercEmLargura (ArvBin *p) {
 Fila f ;
 NoArvBin *q ;
 if (p != NULL) {
   InicializaFila (f) ;
   InsereFila (f, p) ;
   do {
     retira (f, q) ;
     printf (''%d\n'', q->info) ;  /* visita o no */
     if (q->esq != NULL) InsereFila(f, q->esq) ;
     if (q->dir != NULL) InsereFila(f, q->dir) ;
   } while (!FilaVazia(f)) ;
 }
}
```

– O tempo de execução é proporcional ao número de nós $n \rightarrow O(n)$.

- Implementação eficiente de percursos:

1. Os algoritmos de percurso apresentados são eficientes pois o tempo de execução é proporcional ao número de nós da árvore.
2. Podemos melhorar este tempo eliminando a recursão, substituindo-a por uma pilha explícita.
3. A pilha é utilizada para "lembrar" os pontos nos quais a função desce à esquerda ou à direita na árvore, para poder retraçar o caminho de volta.

- Implementação eficiente de percursos:

1. Os algoritmos de percurso apresentados são eficientes pois o tempo de execução é proporcional ao número de nós da árvore.

2. Podemos melhorar este tempo eliminando a recursão, substituindo-a por uma pilha explícita.

3. A pilha é utilizada para "lembrar" os pontos nos quais a função desce à esquerda ou à direita na árvore, para poder retraçar o caminho de volta.

- Implementação eficiente de percursos:

1. Os algoritmos de percurso apresentados são eficientes pois o tempo de execução é proporcional ao número de nós da árvore.

2. Podemos melhorar este tempo eliminando a recursão, substituindo-a por uma pilha explícita.

3. A pilha é utilizada para "lembrar" os pontos nos quais a função desce à esquerda ou à direita na árvore, para poder retraçar o caminho de volta.

- Percurso em pré-ordem iterativo:

```
typedef enum{false, true} boolean

void PreOrdemI( ArvBin *p) {
  Pilha s; boolean fim;
  InicializaPilha(s); fim = false;
  do {
    if(p != NULL) {
      printf(''%d\n'', p->info) ;  /* visita o no */
      if(p->dir != NULL) Empilha(s, p->dir) ;
      p = p->esq ;
    }
    else if(PilhaVazia(s)) fim = true ;
          else Desempilha(s, &p) ;
  } while (!fim) ;
} /*PreOrdemI */
```

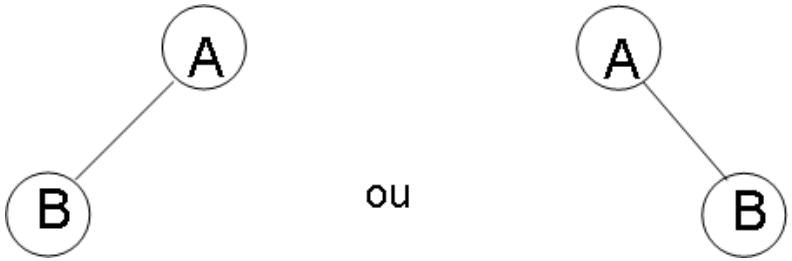
- Percurso em pós-ordem iterativo:

```
void PosOrdemI (ArvBin *p) {
  Pilha s; boolean sobe; int m; InicializaPilha(s) ;
  do {
    while(p != NULL) {
      Empilha(s, p, 1); p = p->esq ;
    }
    sobe = true ;
    while(sobe && !PilhaVazia(s)) {
      Desempilha(s, &p, &m) ;
      switch(m) {
        case 1: Empilha(s, p, 2); p = p->dir;
                sobe = false; break;
        case 2: printf(''%d\n'', p->info); /* visita o no */
      }
    }
  } while(!PilhaVazia(s)) ;
} /* PosOrdemI */
```

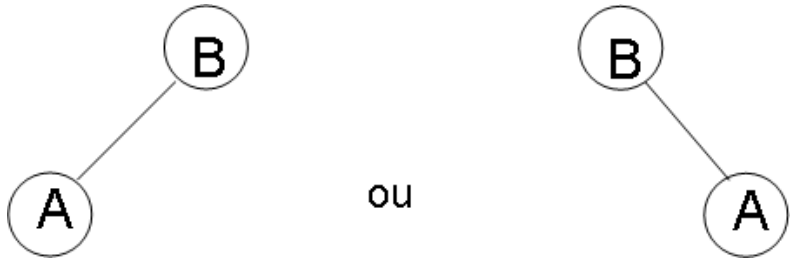
- Percurso em inordem iterativo:

```
void InOrdemI (ArvBin *p) {
  boolean fim; Pilha s; InicializaPilha(s); fim = false;
  do {
    while(p != NULL) {
      Empilha(s, p) ;
      p = p->esq ;
    }
    if(!PilhaVazia(s)) {
      Desempilha(s, &p) ;
      printf(''%d\n'', p->info); /* visita o no */
      p = p->dir ;
    }
    else
      fim = true ;
  } while(!fim) ;
} /* InOrdemI */
```

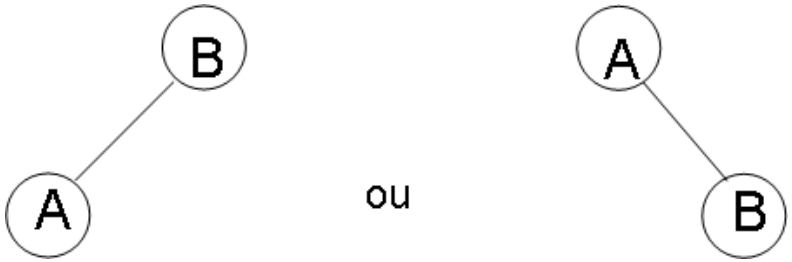
– A árvore percorrida em pré-ordem e que produz a sequência AB é dada por:



– A árvore percorrida em pós-ordem e que produz a sequência AB é dada por:

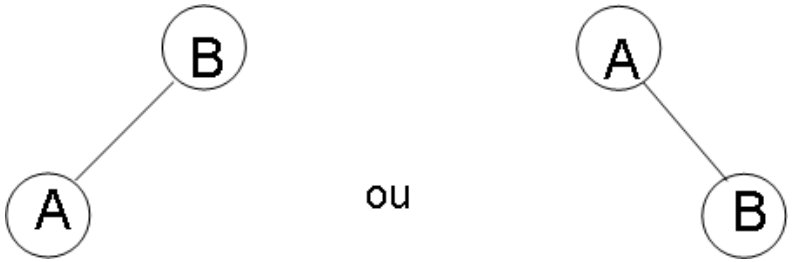


– A árvore percorrida em inordem e que produz a sequência AB é dada por:



• Conclusão: Dada apenas uma ordem de percurso, não se consegue determinar qual a árvore *única* resultante de uma determinada sequência de nós.

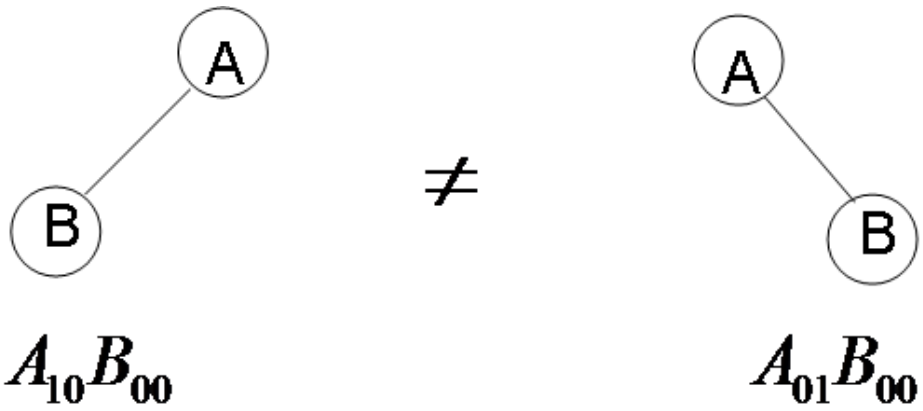
– A árvore percorrida em inordem e que produz a sequência AB é dada por:



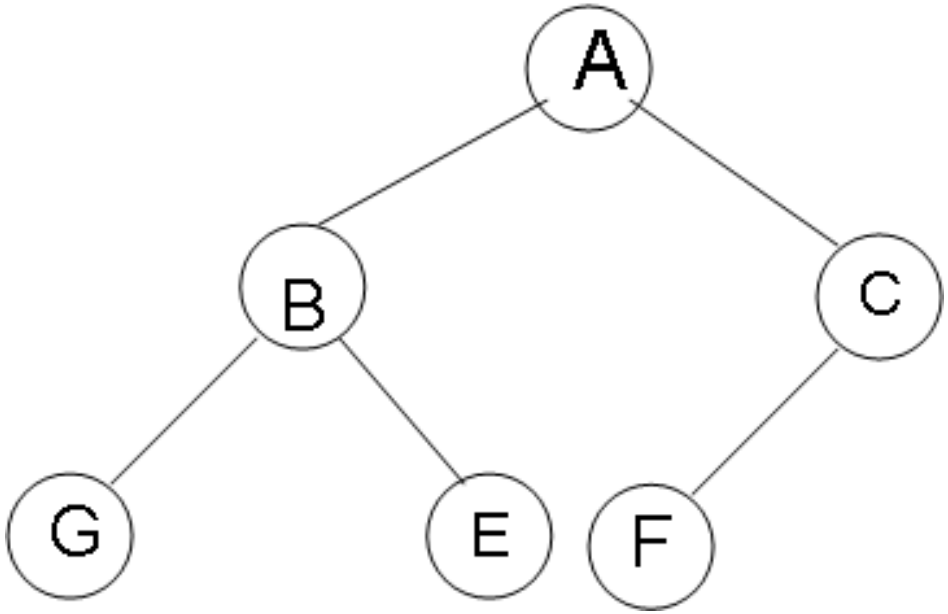
• Conclusão: Dada apenas uma ordem de percurso, não se consegue determinar qual a árvore *única* resultante de uma determinada sequência de nós.

• Alternativa 1: fornecer um percurso mais alguma informação adicional.

• Alternativa 1: fornecer um percurso mais alguma informação adicional.

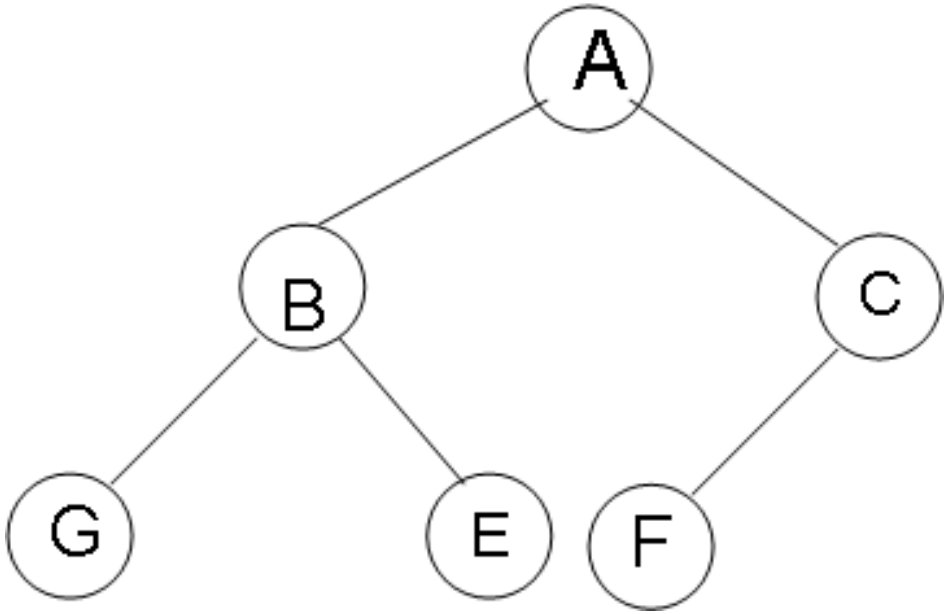


- A sequência $A_{11}B_{11}G_{00}E_{00}C_{10}F_{00}$ em pré-ordem se torna:



– Exercício: Construir a árvore referente à sequência em pré-ordem: $A_{11}B_{11}D_{00}E_{10}G_{00}C_{10}F_{01}H_{00}$

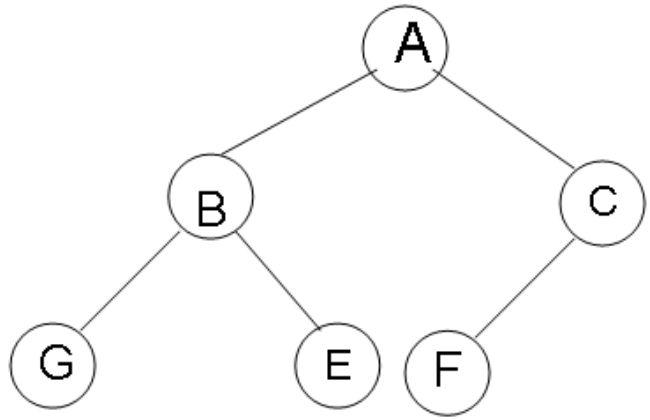
- A sequência $A_{11}\,B_{11}\,G_{00}\,E_{00}\,C_{10}\,F_{00}$ em pré-ordem se torna:



– Exercício: Construir a árvore referente à sequência em pré-ordem: $A_{11}\,B_{11}\,D_{00}\,E_{10}\,G_{00}\,C_{10}\,F_{01}\,H_{00}$

• Alternativa 2: Representação parentética:

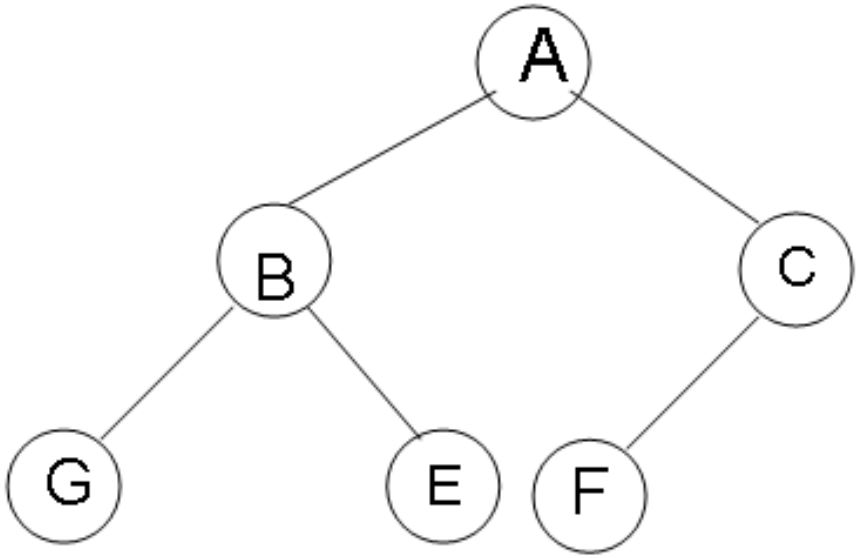
– Caso inordem: Cada nó é descrito por um abre-parêntese, a descrição da sua subárvore esquerda, o rótulo do nó, a descrição da subárvore direita, e por um fecha-parêntese.



((((()G())B(()E()))A(((()F())C())))

• Alternativa 2: Representação parentética:

– Caso pré-ordem: Cada nó é descrito por um abre-parêntese, o rótulo do nó, a descrição da sua subárvore esquerda, a descrição da subárvore direita, e por um fecha-parêntese.

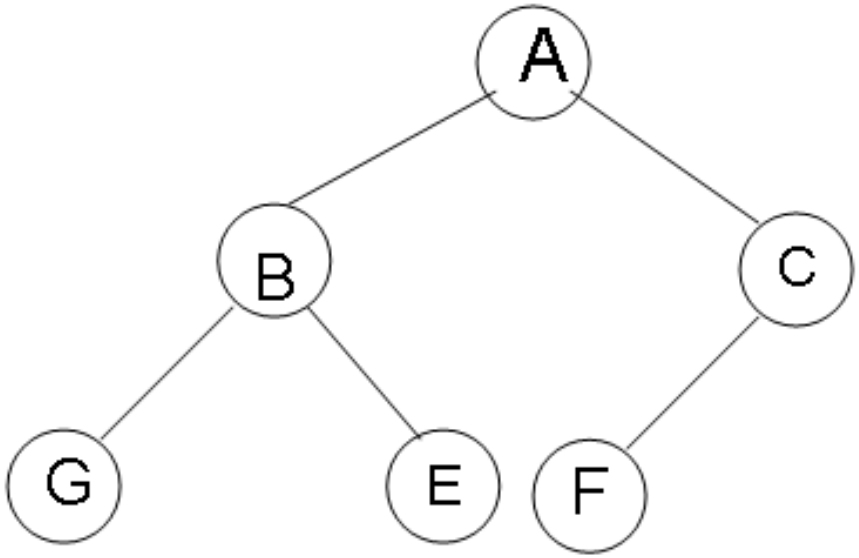


(A(B(G()())(E()()))(C(F()())())))

–Exercício: Definir o caso pós-ordem.

• Alternativa 2: Representação parentética:

– Caso pré-ordem: Cada nó é descrito por um abre-parêntese, o rótulo do nó, a descrição da sua subárvore esquerda, a descrição da subárvore direita, e por um fecha-parêntese.

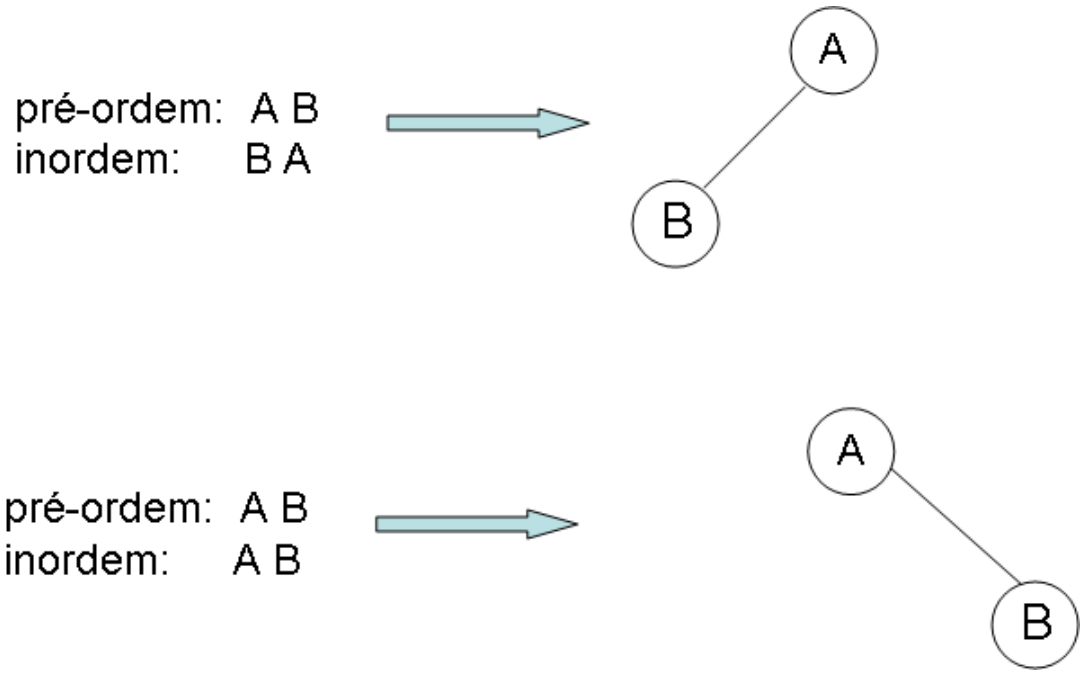


(A(B(G()())(E()()))(C(F()())())))

–Exercício: Definir o caso pós-ordem.

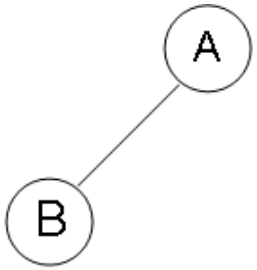
• Alternativa 3: Reconstruir a árvore binária a partir da informação de duas ordens de percurso fornecidas.

Exemplo 1:

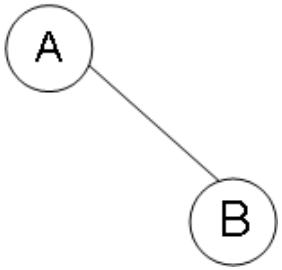


Exemplo 2:

pós-ordem: B A
inordem: B A

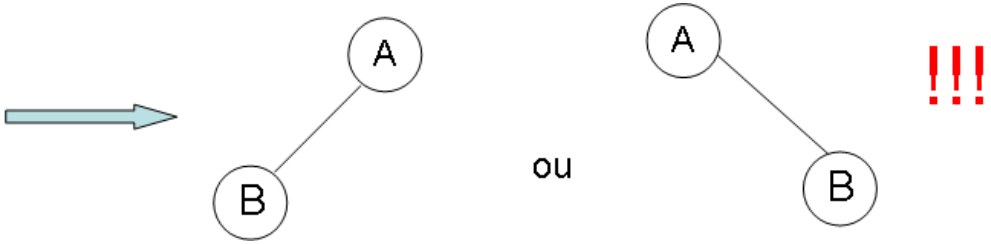


pós-ordem: B A
inordem: A B

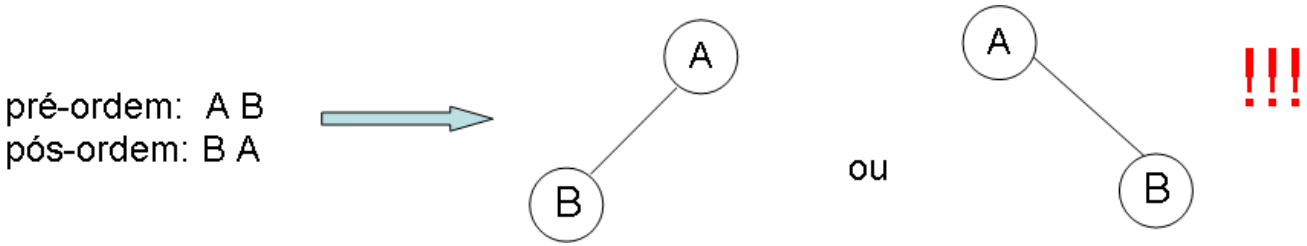


Exemplo 3:

pré-ordem: A B
pós-ordem: B A



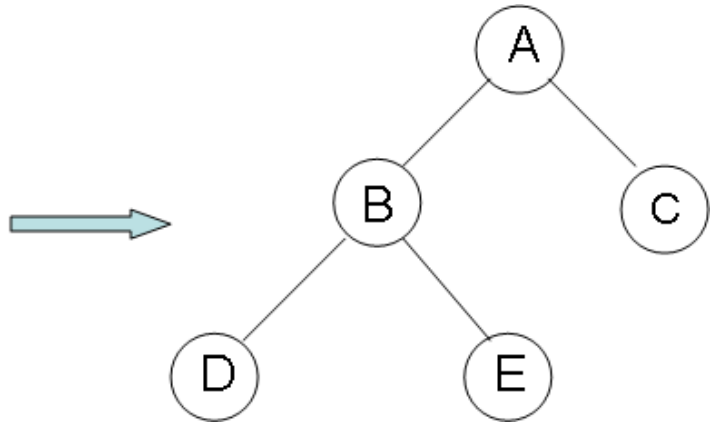
Exemplo 3:



– Conclusão: Não é possível distinguir entre configurações de árvores binárias dados os percursos em pré-ordem e pós-ordem.

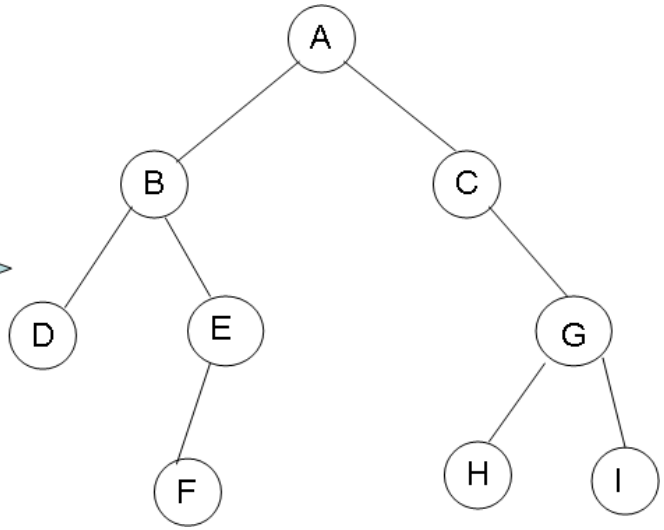
- Exemplo:

pré-ordem: A B D E C
inordem: D B E A C



- Exemplo:

pré-ordem: A B D E F C G H I
inordem: D B F E A C H G I



- Exercício:

– Baseado numa árvore binária, elabore um algoritmo que detecte todas as repetições ocorridas numa lista de números fornecidos sequencialmente.
Exemplo de lista: 14, 15, 4, 9, 7, 18, 3, 5, 16, 4, 20, 17, 9, 14, 5

– Qual a diferença entre se usar uma árvore binária ou uma lista ligada, por exemplo?

- Exercício:

– Baseado numa árvore binária, elabore um algoritmo que detecte todas as repetições ocorridas numa lista de números fornecidos sequencialmente.
Exemplo de lista: 14, 15, 4, 9, 7, 18, 3, 5, 16, 4, 20, 17, 9, 14, 5

– Qual a diferença entre se usar uma árvore binária ou uma lista ligada, por exemplo?

- Exercício:

Escreva uma função que receba uma árvore binária não vazia e devolva o endereço do primeiro nó da árvore percorrida em ordem simétrica (inordem). Escreva outra função que devolva o último nó da árvore.