Árvores e Árvores Binárias

Siang Wun Song - Universidade de São Paulo - IME/USP

MAC 5710 - Estruturas de Dados - 2008

Referência bibliográfica

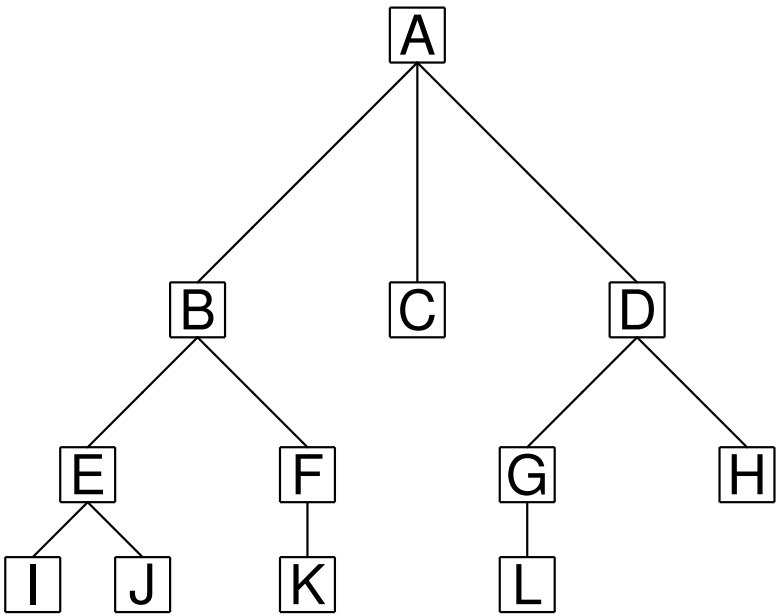
Os slides sobre este assunto são parcialmente baseados nas seções sobre árvores do capítulo 4 do livro

- N. Wirth. Algorithms + Data Structures = Programs. Prentice Hall, 1976.

Árvore

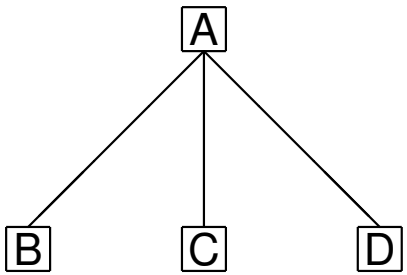
Uma árvore do tipo T é constituída de

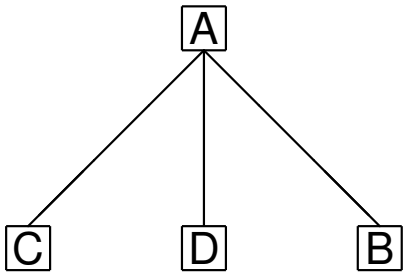
- uma estrutura vazia, ou
- um elemento ou um nó do tipo T chamado raiz com um número finito de árvores do tipo T associadas, chamdadas as sub-árvores da raiz.



Nomenclatura: árvore ordenada

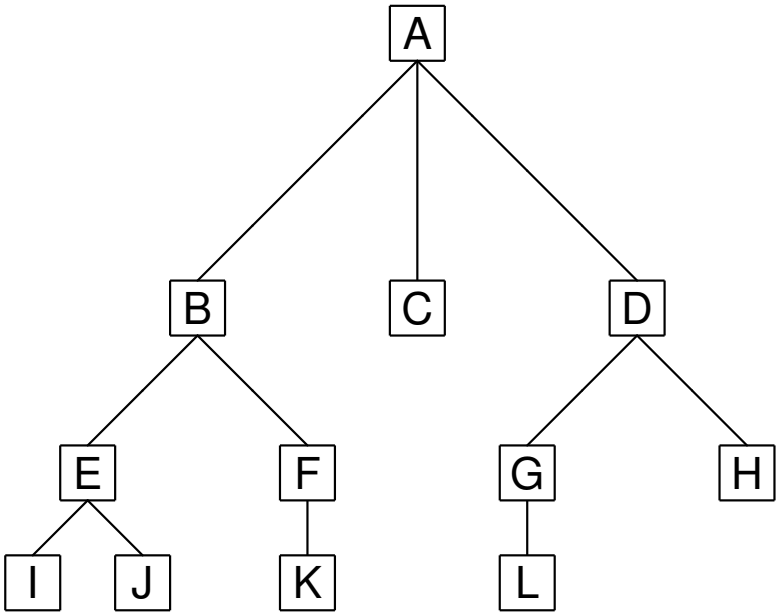
Uma árvore é chamada ordenada quando a ordem das subárvores é significante. Assim, as duas árvores ordenadas seguintes são diferentes.





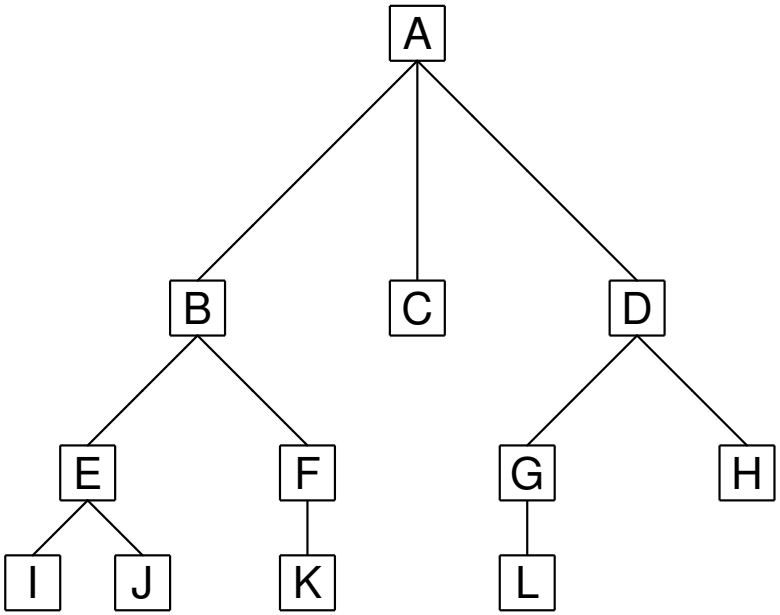
- Numa árvore que representa os descendentes de uma família real, a ordem das subárvores pode ser importante pois pode determinar a ordem de sucessão da coroa.

Nomenclatura: pai, filho, nível



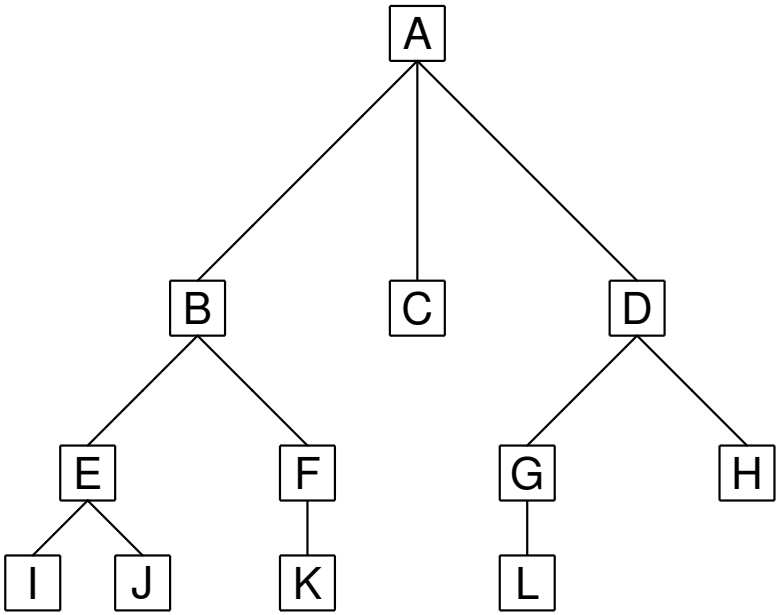
- Pai e filho: Um nó y abaixo de um nó x é chamado filho de x. x é dito pai de y. Exemplo: B é pai de E e F.
- Irmão: Nós com o mesmo pai são ditos irmãos. Exemplo: B, C, D são irmãos.
- Nível de um nó: A raiz de uma árvore tem nível 1. Se um nó tem nível i, seus filhos têm nível $i + 1$. Exemplo: E, F, G e H têm nível 3.

Nomenclatura: altura, folha, grau



- Altura ou profundidade de uma árvore: É o máximo nível de seus nós. A árvore do exemplo tem altura 4.
- Folha ou nó terminal: É um nó que não tem filhos. Exemplo: I, J, K, L são folhas.
- Nó interno ou nó não terminal: É um nó que não é folha.
- Grau de um nó: É o número de filhos do nó. Exemplo: B tem grau 2, G tem grau 1.

Nomenclatura: grau de árvore, árvore binária



- Grau de uma árvore: É o máximo grau de seus nós. A árvore do exemplo tem grau 3.

Usando a nomenclatura vista, podemos definir a árvore binária.

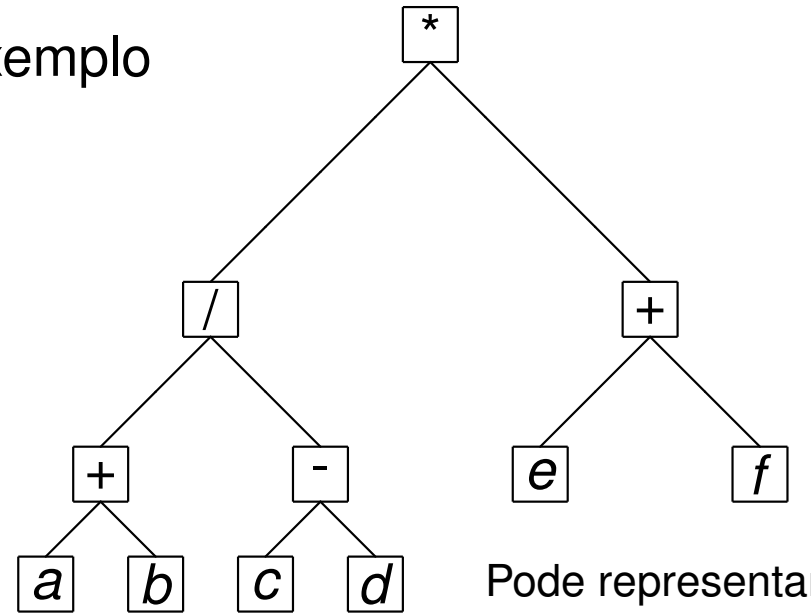
Árvore binária: É uma árvore ordenada de grau 2.

Árvore binária

Uma árvore binária é

- vazia ou
- um nó chamado raiz mais duas árvores binárias disjuntas chamadas subárvore esquerda e subárvore direita.

Exemplo



Pode representar a expressão aritmática:

$$((a+b)\ /(c-d))\ *(e\ +\ f)$$

Veja como a estrutura de árvore binária expressa de maneira clara a precedência das operações, sem necessidade de usar parêntesis.

Aplicações que usam árvores e árvore binárias

- Problemas de busca de dados armazenados na memória principal do computador: árvore binária de busca, árvores (quase) balanceadas como AVL, rubro-negra, etc.
- Problemas de busca de dados armazenados na memória secundárias principal do computador (disco rígico): e.g. B-árvores.
- Aplicações em Inteligência Artificial: árvores que representam o espaço de soluções, e.g. jogo de xadrez, resolução de problemas, etc.
- No processamento de cadeias de caracteres: árvore de sufixos.
- Na gramática formal: árvore de análise sintática.
- Em problemas onde a meta é achar uma ordem que satisfaz certas restrições (e.g. testar a propriedade de uns-consecutivos numa matriz, reconhecer grafos intervalo, planaridade de grafo, etc.): árvore-PQ.

Implementação de uma árvore binária

Em Pascal podemos declarar o registro node assim:

```
type node = record
      key: integer;
      left, right: ∧node
      end
```

Vamos fazer um pequeno exercício: construir uma árvore binária constituída de n nós, para um dado n, que tenha mínima altura.

Para isso, vamos alocar o máximo numero possível de nós em cada nível da árvore, exceto no último que pode estar incompleto.

Podemos distribuir nós em igual quantidade para a esquerda e a direita de cada nó. Teremos uma árvore perfeitamente balanceada.

Construção de uma árvore perfeitamente balanceada

Árvore perfeitamente balanceada: para todo nó da árvore, os números de nós das suas duas subárvores diferem no máximo em um.

Seja a função *tree*(n) que gera uma árvore perfeitamente balanceada com n nós.

Informalmente *tree*(n) pode ser definida recursivamente assim:

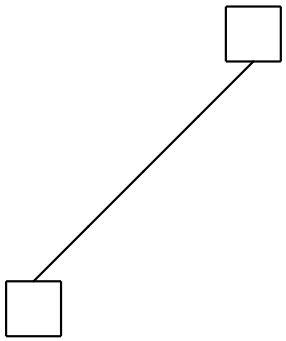
1. Aloque um nó para ser a raiz.
2. Coloque na esquerda da raiz uma árvore gereada por *tree*($nl = n$ div 2).
3. Coloque na direita da raiz uma árvore gereada por *tree*($nr = n - n_l - 1$).

Veremos exemplos de árvores assim construídas.

Árvore perfeitamente balanceada com $n = 1$

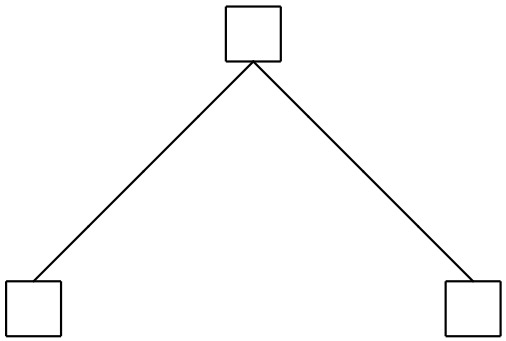
$n = 1$

Árvore perfeitamente balanceada com $n = 2$



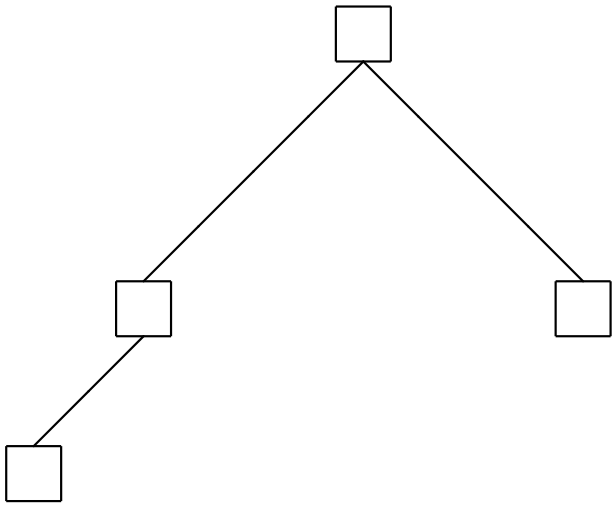
$n = 2$

Árvore perfeitamente balanceada com $n = 3$



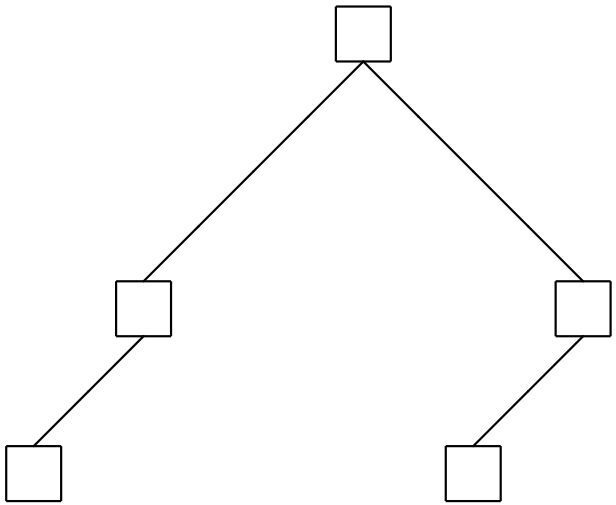
$n = 3$

Árvore perfeitamente balanceada com $n = 4$



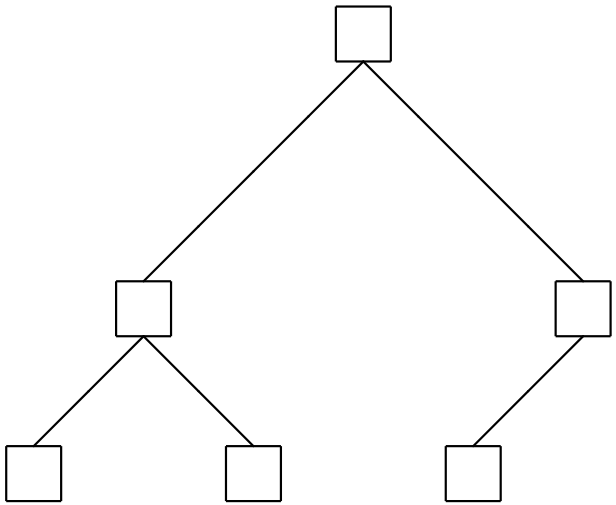
$n = 4$

Árvore perfeitamente balanceada com $n = 5$



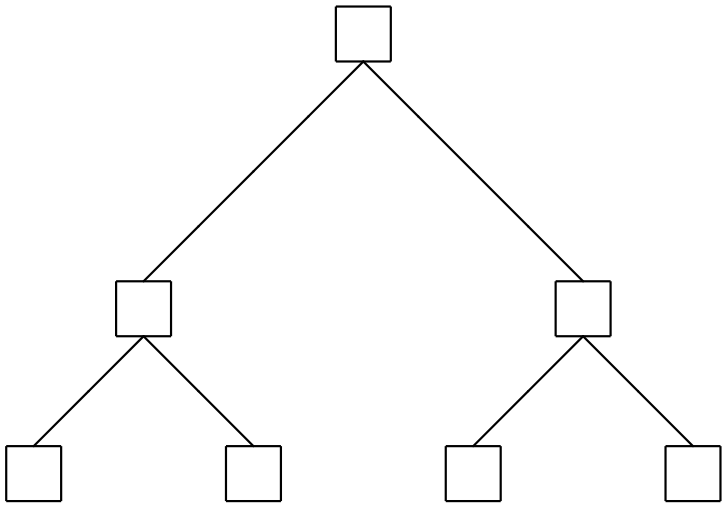
$n = 5$

Árvore perfeitamente balanceada com $n = 6$



$n = 6$

Árvore perfeitamente balanceada com $n = 7$



$n = 7$

Progama em Pascal - livro de N. Wirth

```
type ref = ∧node;
     node = record
          key: integer;
          left, right: ref
          end;
var n: integer; root: ref;
function tree(n: integer): ref;
var newnode: ref;
     x, nl, nr: integer;
begin { constrói uma árvore perf. balanceada de n nós}
if n = 0 then
     tree:=nil
else
     begin
          nl:= n div 2;
          nr:= n - nl - 1;
          read(x);
          new(newnode);
          begin
               newnode∧.key:=x;
               newnode∧.left:=tree(nl);
               newnode∧.right:=tree(nr)
          end;
     tree:=newnode
     end
end
```

Formas de percorrer uma árvore

Em algumas aplicações, é necessário percorrer uma árvore de forma sistemática, visitando cada nó da árvore uma única vez, em determinada ordem.

Por exemplo, se cada nó da árvore possui um campo que armazena o salário, então podemos querer visitar cada nó para fazer um reajuste salarial. A visita seria atualizar o campo salário. Não podemos esquecer nenhum nó, nem queremos visitar um nó mais do que uma vez. Neste caso, a ordem de visita não é importante. Mas em algumas outras aplicações, queremos visitar os nós em certa ordem desejada.
Veremos várias formas para percorrer uma árvore binária.

- Pré-ordem.
- In-ordem ou ordem simétrica.
- Pós-ordem.

Percorrer uma árvore binária em pré-ordem

Percorrer uma árvore binária em pré-ordem:

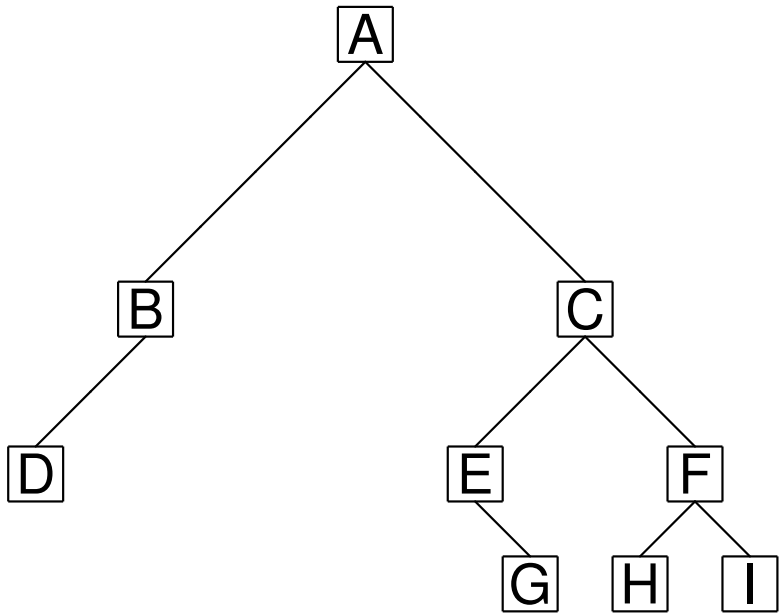
1. Vistar a raiz.
2. Percorrer a sua subárvore esquerda em pré-ordem.
3. Percorrer a sua subárvore direita em pré-ordem.

Visitar um nó significa executar uma certa ação no nó.

Exemplo de percurso em pré-ordem

Percorrer uma árvore binária em pré-ordem:

1. Vistar a raiz.
2. Percorrer a sua subárvore esquerda em pré-ordem.
3. Percorrer a sua subárvore direita em pré-ordem.



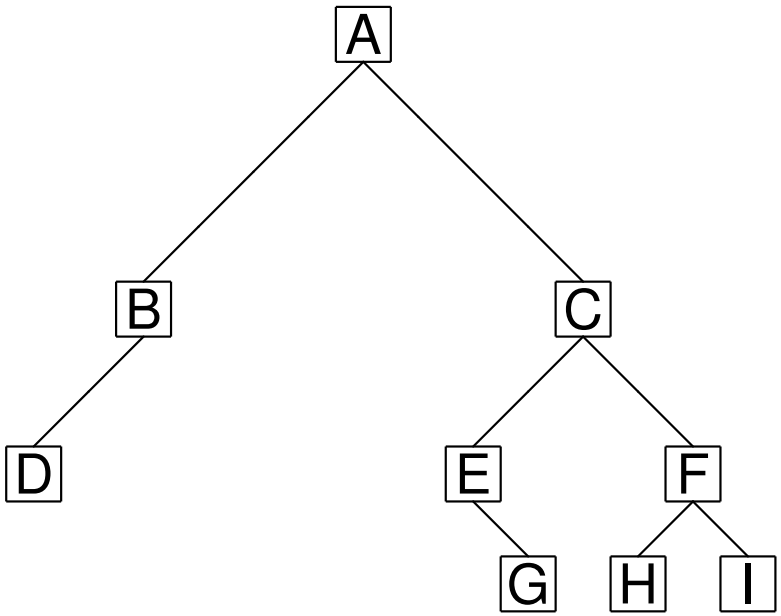
Percurso em pré-ordem: A B D C E G F H I

O percurso em pré-ordem segue os nós até chegar os mais “profundos”, em “ramos” de subárvores da esquerda para a direita. É conhecida usualmente pelo nome de percurso em profundidade (*depth-first*).

Exemplo de percurso em pré-ordem

Percorrer uma árvore binária em pré-ordem:

1. Vistar a raiz.
2. Percorrer a sua subárvore esquerda em pré-ordem.
3. Percorrer a sua subárvore direita em pré-ordem.



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

O percurso em pré-ordem segue os nós até chegar os mais “profundos”, em “ramos” de subárvores da esquerda para a direita. É conhecida usualmente pelo nome de percurso em profundidade (*depth-first*).

Procedimento pre-order em Pascal

É fácil escrever um procedimento recursivo para realizar a pré-ordem. Em Pascal:

```
type ref = ∧node;
     node = record
          key: integer;
          left, right: ref
          end;
procedure pre-order(t: ref);
begin
     if t ≠ nil then
     begin
          visita(t);
          pre-order(t∧.left);
          pre-order(t∧.right);
     end
end
```

Percorrer uma árvore binária em in-ordem

Percorrer uma árvore binária em in-ordem:

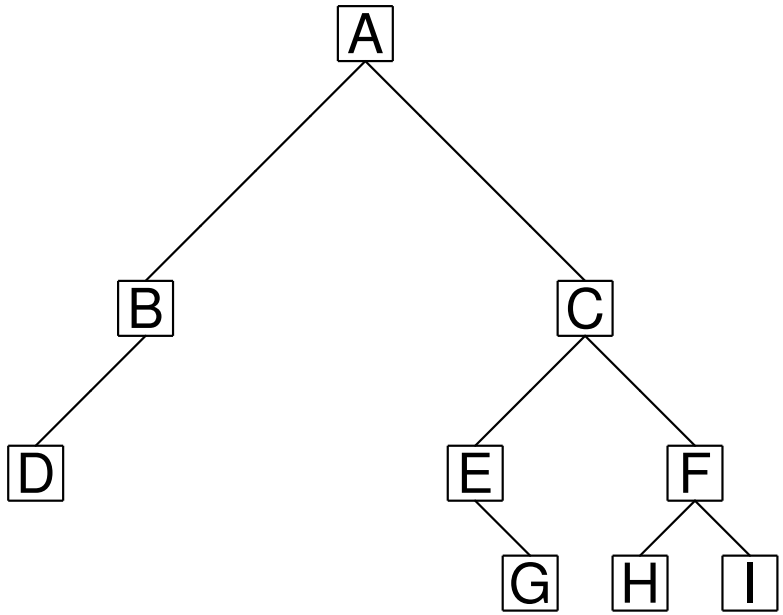
1. Percorrer a sua subárvore esquerda em in-ordem.
2. Vistar a raiz.
3. Percorrer a sua subárvore direita em in-ordem.

A in-ordem visita a raiz entre as ações de percorrer as duas subárvores. É conhecida também pelo nome de ordem simétrica.

Exemplo de percurso em in-ordem

Percorrer uma árvore binária em in-ordem:

1. Percorrer a sua subárvore esquerda em in-ordem.
2. Vistar a raiz.
3. Percorrer a sua subárvore direita em in-ordem.



Percurso em in-ordem: D B A E G C H F I

Procedimento in-order em Pascal

É fácil escrever um procedimento recursivo para realizar a in-ordem. Em Pascal:

```
type ref = ∧node;
     node = record
          key: integer;
          left, right: ref
          end;
procedure in-order(t: ref);
begin
     if t ≠ nil then
     begin
          in-order(t∧.left);
          visita(t);
          in-order(t∧.right);
     end
end
```

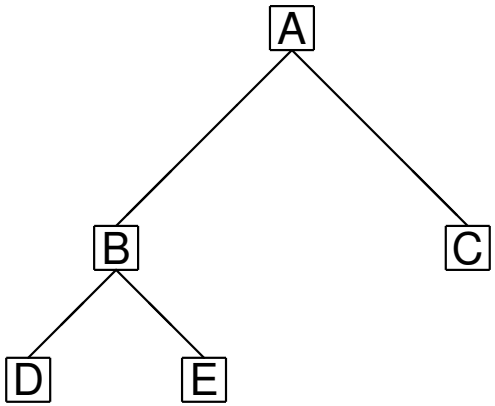
Exemplo de percurso em in-ordem

Queremos imprimir os nós de uma árvore binária apontada por t em uma impressora do tipo matricial, em que cada linha é impressa de cada vez. A visita de um nó significa imprimi-lo pela impressora.

A ordem de visita dos nós é quase a in-ordem, que chamaremos de in'-ordem, com a simples inversão na ordem de percurso das duas subárvores:

Percorrer uma árvore binária em in'-ordem:

1. Percorrer a sua subárvore direita em in'-ordem.
2. Vistar a raiz.
3. Percorrer a sua subárvore esquerda em in'-ordem.



Árvore impressa:

C

A

E

B

D

Procedimento de impressão em Pascal

O livro de Wirth apresenta o seguinte procedimento para imprimir uma árvore binária. Note a sua semelhança com a in-ordem, trocando-se apenas left com right.

```
type ref = ∧node;
     node = record
          key: integer;
          left, right: ref
          end;
procedure print-tree(t: ref; h: integer);
var i: integer;
begin {imprime árvore t com indentação h}
     if t ≠ nil then
     begin
          print-tree(t∧.right, h+1);
          for i:= 1 to h do write('    ');
          writeln(t∧.key);
          print-tree(t∧.left, h+1);
     end
end
```

Percorrer em in-ordem sem usar recursão

A recursão facilita a escrita do algoritmo de percurso, mas podemos fazer o percurso, digamos em in-ordem, sem usar recursão.

Para isso usamos uma pilha. Suponhamos que já existem as rotinas push e pop. O apontador t aponta para a árvore binária a ser percorrida.

in-order(t):

```
 1: p ← t
 2: while p ≠ nil or pilha não vazia do
 3:     if p ≠ nil then
 4:         push(p)
 5:         p ← left(p)
 6:     else
 7:         pop(p)
 8:         visita(p)
 9:         p ← right(p)
10:     end if
11: end while
```

Percorrer uma árvore binária em pós-ordem

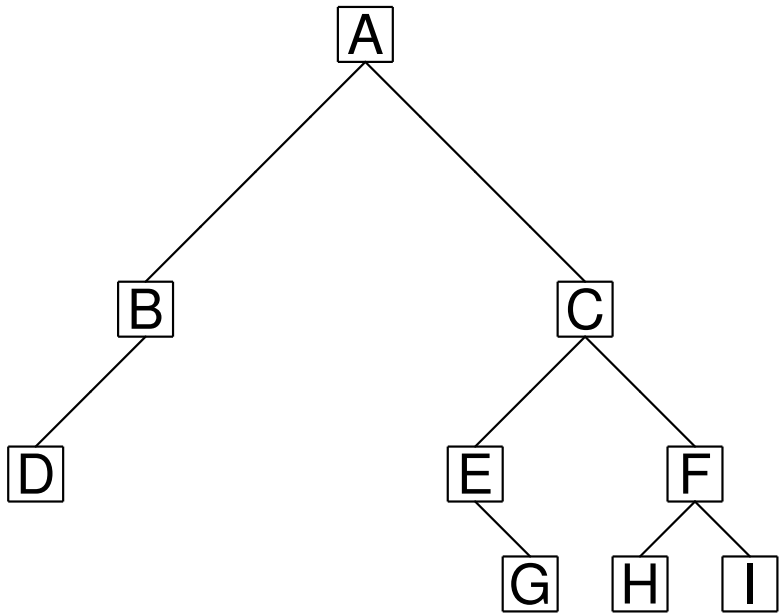
Percorrer uma árvore binária em pós-ordem:

1. Percorrer a sua subárvore esquerda em pós-ordem.
2. Percorrer a sua subárvore direita em pós-ordem.
3. Vistar a raiz.

Exemplo de percurso em pós-ordem

Percorrer uma árvore binária em pós-ordem:

1. Percorrer a sua subárvore esquerda em pós-ordem.
2. Percorrer a sua subárvore direita em pós-ordem.
3. Vistar a raiz.



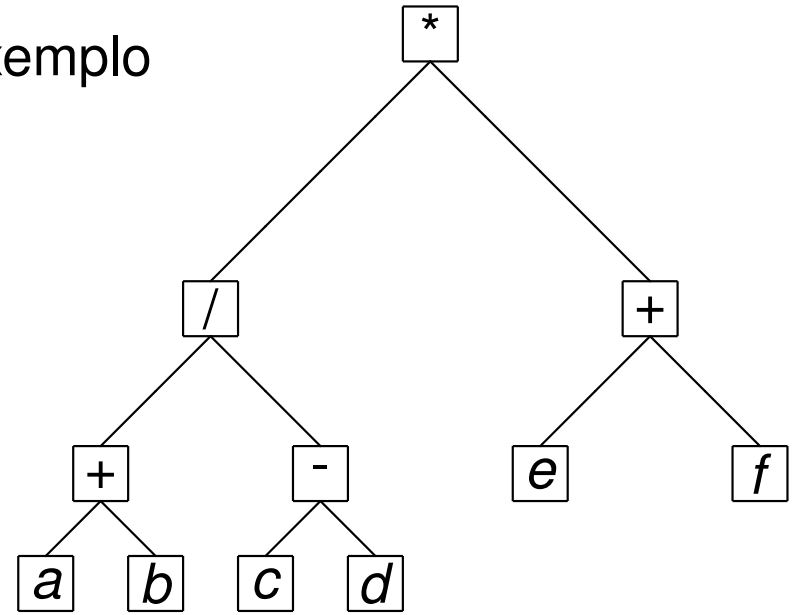
Percurso em pós-ordem: D B G E H I F C A

Outro exemplo de percurso em pós-ordem

Percorrer uma árvore binária em pós-ordem:

1. Percorrer a sua subárvore esquerda em pós-ordem.
2. Percorrer a sua subárvore direita em pós-ordem.
3. Vistar a raiz.

Exemplo



Percurso em pós-ordem: $a\ b + c\ d - /\ e\ f + *$

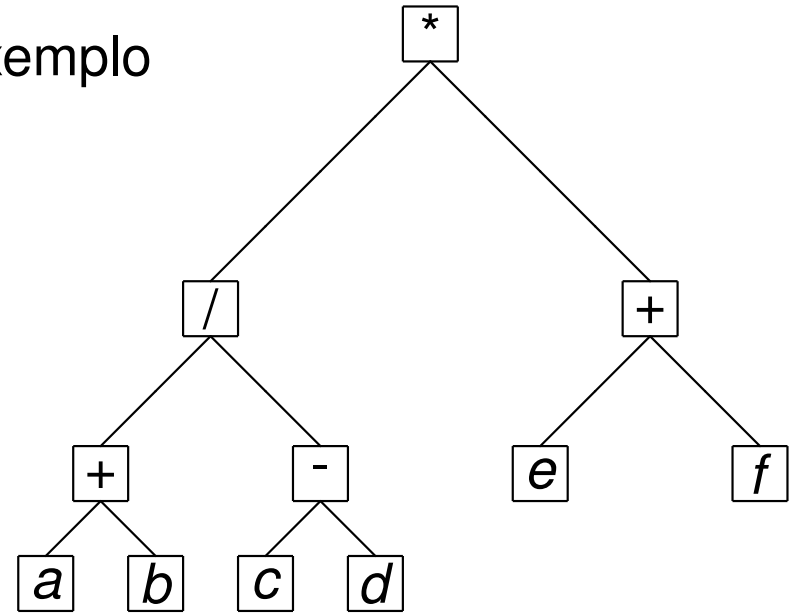
A representação de uma expressão aritmética com o operador no final dos operandos é conhecida pelo nome de notação reversa ou polonesa.

Outro exemplo de percurso em pós-ordem

Percorrer uma árvore binária em pós-ordem:

1. Percorrer a sua subárvore esquerda em pós-ordem.
2. Percorrer a sua subárvore direita em pós-ordem.
3. Vistar a raiz.

Exemplo



Percurso em pós-ordem: $a\ b + c\ d - /\ e\ f + *$

A representação de uma expressão aritmética com o operador no final dos operandos é conhecida pelo nome de notação reversa ou polonesa.

Procedimento post-order em Pascal

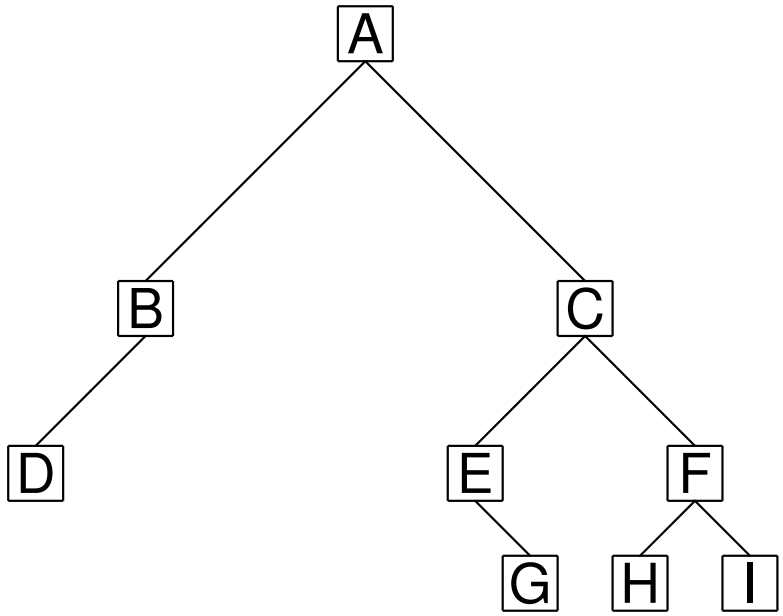
É fácil escrever um procedimento recursivo para realizar a pós-ordem. Em Pascal:

```
type ref = ∧node;
     node = record
          key: integer;
          left, right: ref
          end;
procedure post-order(t: ref);
begin
     if t ≠ nil then
     begin
          post-order(t∧.left);
          post-order(t∧.right);
          visita(t);
     end
end
```

Percurso em largura

Há ainda outras formas usuais para percorrer uma árvore.

Percurso em largura ou em nível (*breadth-first*): percorre a árvore em ordem crescente de seus níveis e, em cada nível, da esquerda para a direita. Uma fila é usada para realizar este percurso.



Percurso em largura: A B C D E F G H I

Aplicações que usam percurso de árvores

Há várias aplicações que se baseiam em percurso de árvores.

- Adota-se uma ordem apropriada de percurso.
- Usa-se uma rotina adequada para visitar cada nó.

Exercícios:

1. Considere uma árvore binária dada onde cada nó tem um campo adicional chamado *alt*. Escreva um algoritmo que coloque no campo *alt* de cada nó *x* da árvore binária a altura da subárvore enraizada em *x*. Dica: use pós-ordem e uma rotina apropriada de visita. Tente fazer este exercício e veja o por que da adoção da pós-ordem.
2. Em algumas situações, é desejável ter um campo adicional *pai* em cada nó da árvore que aponta para o nó pai. No caso de raiz, o seu campo pai deve apontar para nil.
   Suponha uma árvore binária existente que reservou este campo *pai* em cada nó mas apenas os campos *key*, *left* e *right* estão definidos. Escreva uma algoritmo para definir o