ESTRUTURA DE DADOS

Árvores, árvores binárias e percursos

Cristina Boeres

Árvores

- utilizada em muitas aplicações
- modela uma hierarquia entre elementos
  - árvore genealógica
  - diagrama hierárquico de uma organização
  - modelagem de algoritmos
- O conceito de árvores está diretamente ligado à recursão

Árvores

- um conjunto finito de elementos onde
  - um elemento é chamado de raiz
  - os outros são divididos em subconjuntos disjuntos, onde cada um define uma árvore
    - cada elemento é um *nó* ou *vértice* da árvore
    - arcos ou arestas conectam os vértices

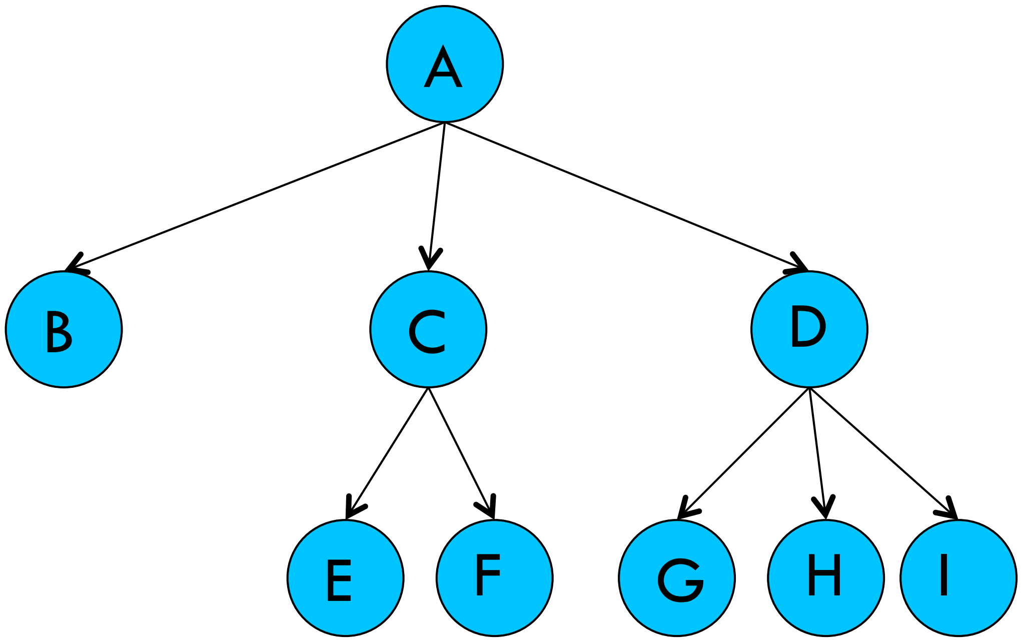
Árvores

- uma coleção não vazia de vértices e *ramos* que satisfazem a certos requisitos

- vértice (ou nó):
  - é um objeto simples que pode ter um nome e mais alguma outra informação associada

- arco ou aresta (direcionado ou não):
  - é uma conexão entre dois nós

Árvores - Representação



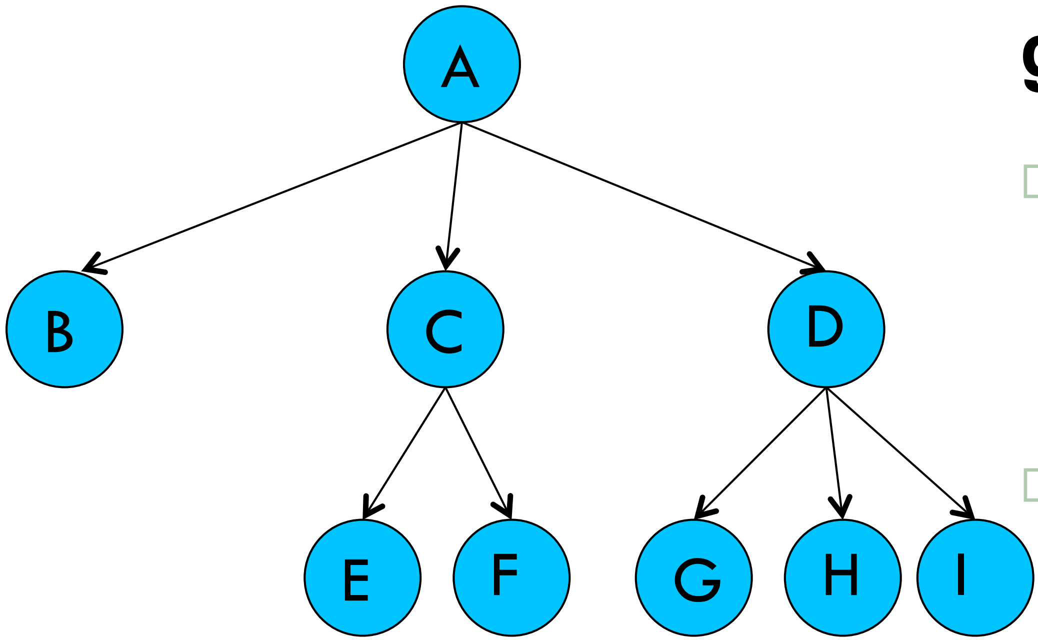
Terminologia e Propriedades

- cada vértice (exceto a raiz) tem exatamente um antecessor imediato ou pai
- cada vértice tem nós sucessores imediatos ou filhos, a não ser:
  - nós sem filhos → terminais ou folhas
- filhos de um mesmo pai : irmãos
- nós com pelo menos um filho:
  - não-terminais ou internos

Terminologia e Propriedades

- caminho em uma árvore:
  - é uma lista de vértices distintos e sucessivos, conectados por arcos (arestas) da árvore
- nó raiz
  - existe exatamente um caminho entre a raiz e cada um dos nós da árvore
    - se existir mais de um caminho ou nenhum: grafo

Terminologia e Propriedades



grau de um vértice

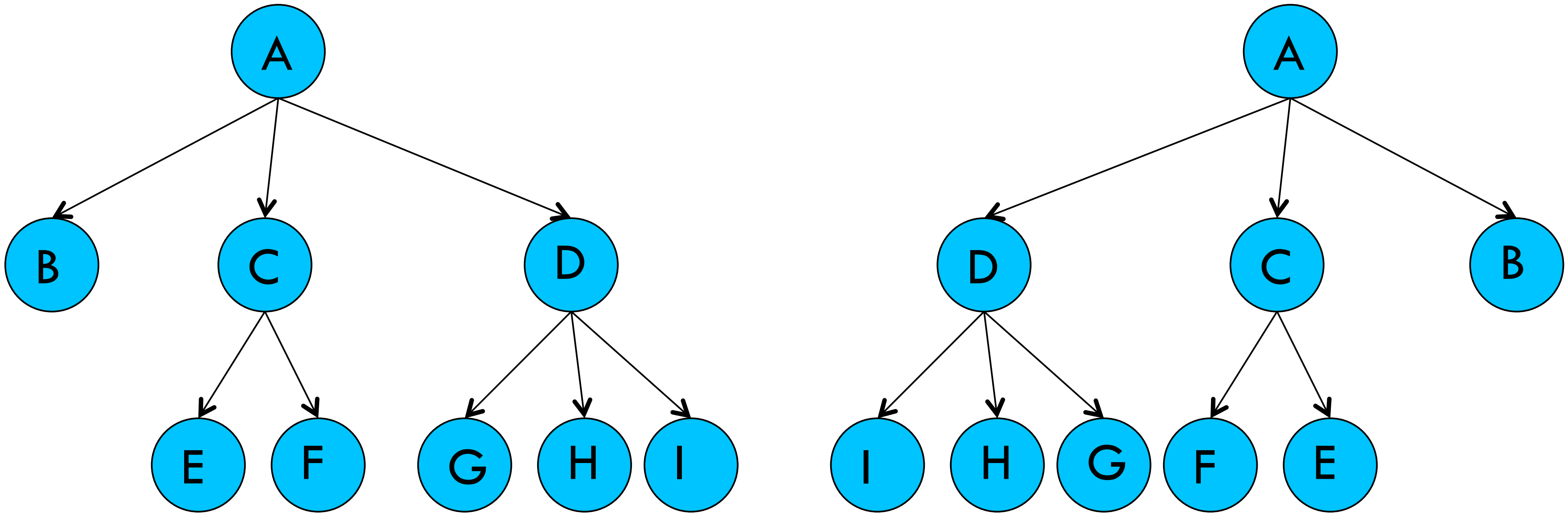
- é o número de subárvores não vazias de um nó
- no exemplo
  - grau de A = 3
  - grau de C = 2

Terminologia e Propriedades

- qualquer nó é a raiz de uma sub-árvore consistindo dele e dos nós abaixo

Árvores

- qual a diferença entre as duas árvores?

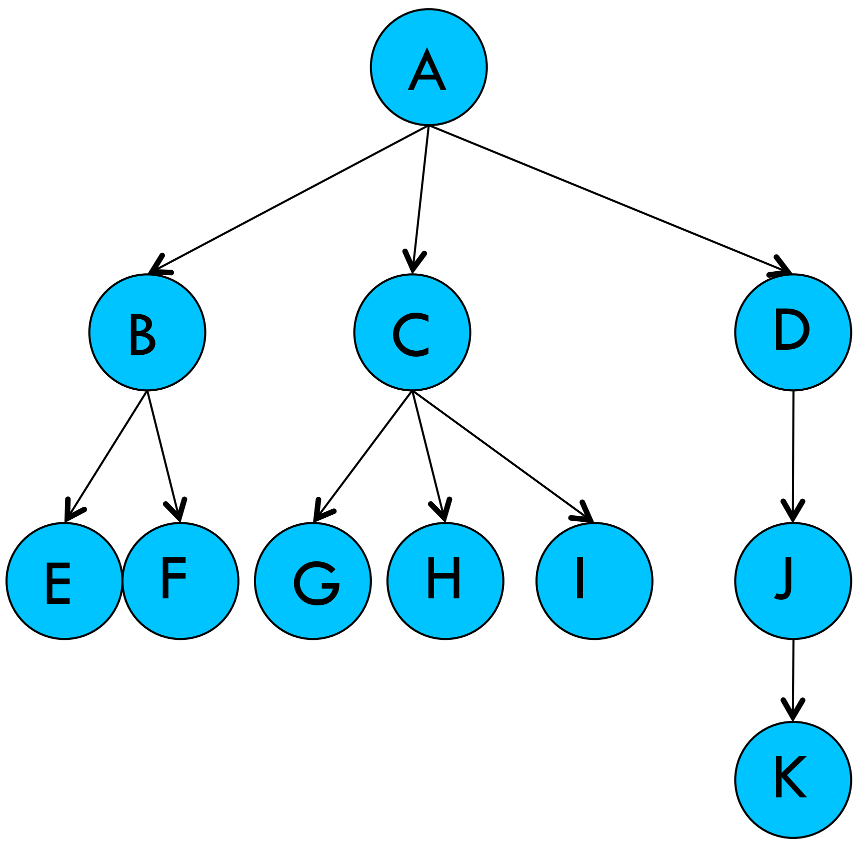


Árvores

- A única diferença entre as duas árvores é a ordem das sub-árvores
  - Uma árvore ordenada é definida como uma árvore onde as sub-árvore formam um conjunto ordenado
  - Em uma árvore ordenada define-se o primeiro, segundo e último irmão, de acordo com alguma propriedade

Terminologia

- os vértices da árvore estão classificados em níveis
  - é o número de nós no caminho entre o vértice e a raiz



⇒ nível da raiz é zero

⇒ nível de C é 1

⇒ nível de K é 3

⇒ **Nível de um nó:**

⇒ nível de seu pai + 1

Terminologia

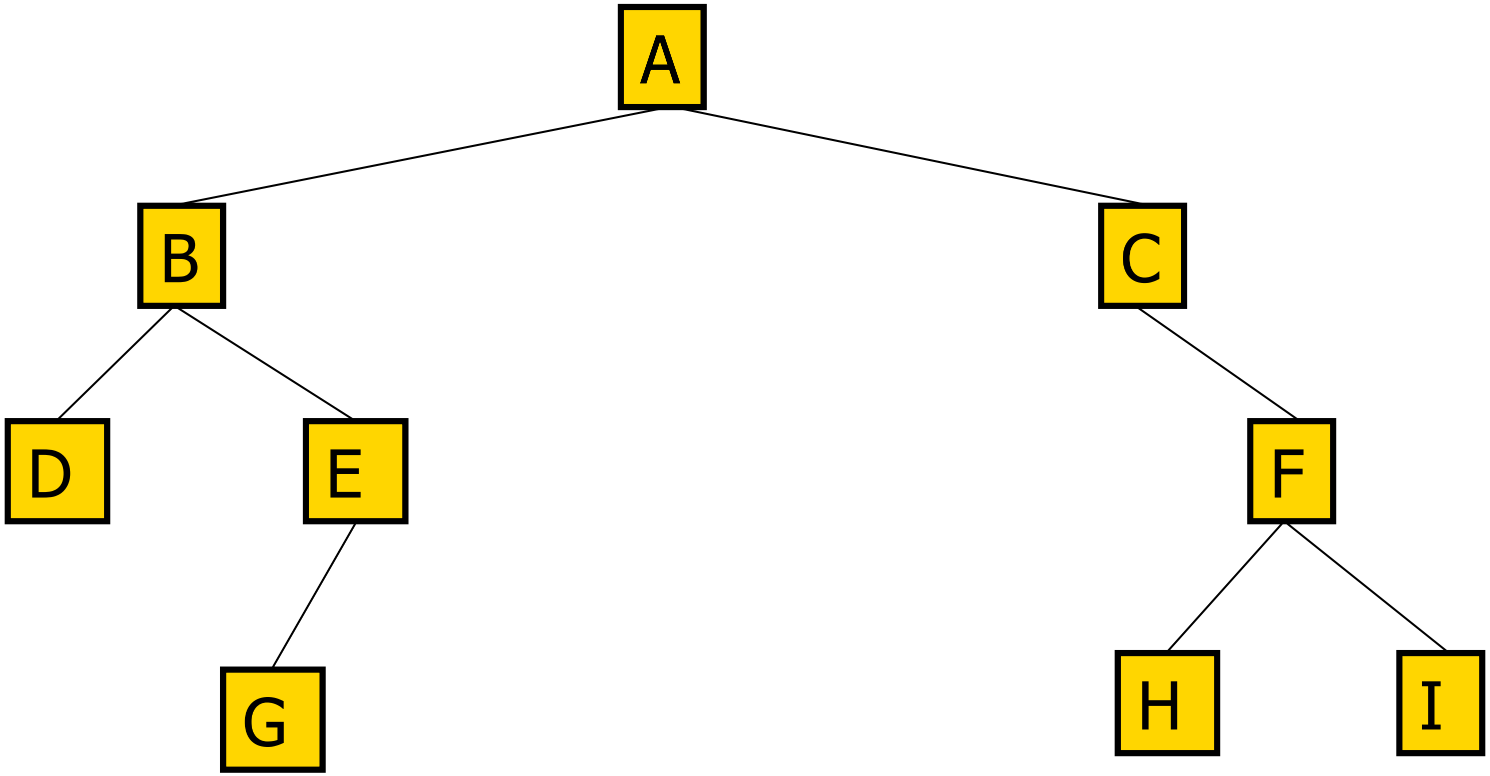
- Altura de uma árvore
  - corresponde ao maior nível
  - maior distância entre a raiz e qualquer nó

- Floresta
  - um conjunto de árvores
  - se removemos a raiz e os arcos que a ligam às sub-árvores, ficamos com uma floresta

Árvore Binária

- é um conjunto finito de elementos que é ou vazio ou composto de três conjuntos disjuntos
  - o primeiro contém um único elemento, a raiz
  - os outros dois subconjuntos são árvores binárias
    - as sub-árvores da esquerda e da direita
    - As sub-árvores da esquerda ou da direita podem estar vazias

Árvore Binária



Árvores Binárias

- considerando que os dois filhos de cada nó interno são ordenados:
  - o filho da esquerda e
  - o filho da direita
  - Cada nó interno tem que ter filho da direita ou da esquerda, sendo que um ou ambos podem ser nós externos

Árvores Binárias

- uma árvore binária vazia:
  - consiste de nenhum nó interno e um nó externo
- uma árvore binária é uma árvore ordenada, na qual cada nó tem 0, 1, ou 2 filhos
  - cada filho corresponde a uma árvore binária

Árvores Binárias

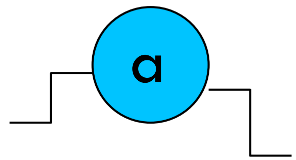
O número de sub-árvores vazias a esquerda ou a direita em uma árvore binária com *n* nós é:

$$n+1$$

Árvores Binárias

Demonstração: explorando a definição recursiva de árvores

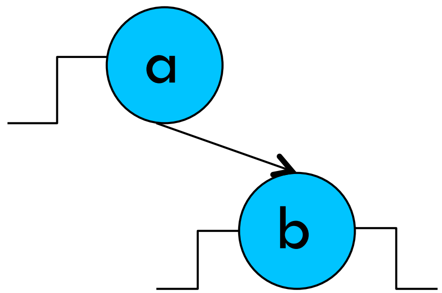
- se n = 1 então 2 subárvores vazias



Árvores Binárias

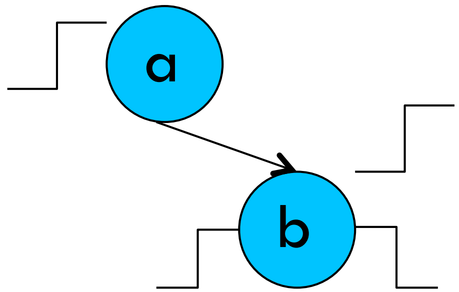
Demonstração: explorando a definição recursiva de árvores

- se n = 1 então 2 subárvores vazias
- se n = 2 então 3 subárvores vazias

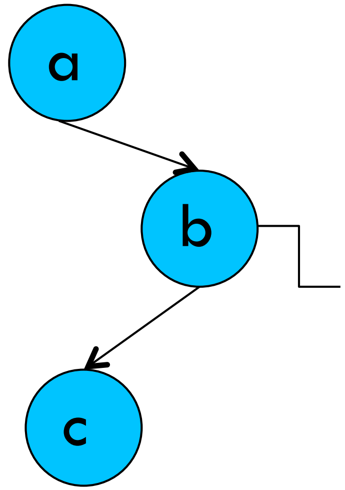


Árvores Binárias

- Supondo que seja verdade para uma Árvore Binária com $n - 1$ vértices
  - ela tem n subárvores vazias



- Então em em uma árvore com n nós
  - uma subárvore vazia foi substituído por um vértice interno e 2 subárvores vazias



  - o número de subárvores vazias é então:
    $n - 1 \;\; +2 \;\; = n + 1$

Árvores Binárias

- Definição de Árvores
  - uma árvore é um único nó ou um nó raiz conectado a um conjunto de árvores

- Definição de Árvores Binárias
  - uma árvore binária é um nó externo (folha) ou um nó raiz (interno) conectado a esquerda e/ou a direita a árvores binárias

Árvore Binária Estrita

- todo nó não folha possui filhos a esquerda e a direita → *Árvore Binária Estrita*

Toda árvore binária estrita com n folhas contém $2n-1$ nós

Árvore Binária Estrita

- árvore binária estrita com:
  - 1 folha: um nó
  - 2 folhas: 3 nós
- hipótese: n folhas → 2n-1 nós
  - adiciona mais 1 folha (total então são n+1 folhas então), pois:
    - para continuar estrita, um vértice folha passa a ter mais acrescentar dois filhos
      - dois novos nós são folhas
      - antigo nós folha passa a ser interno
  - Total de nós: 2n-1 +2 = 2n+1 = 2(n+1)-1

Árvore Binária

- Nível:
  - A raiz tem nível 0
  - A raiz de outro nó é o nível do nó pai +1.
- A profundidade de uma árvore é o maior nível para todas as folhas
  - Markeson: raiz tem nível 1
  - Langsan : raiz tem nível 0

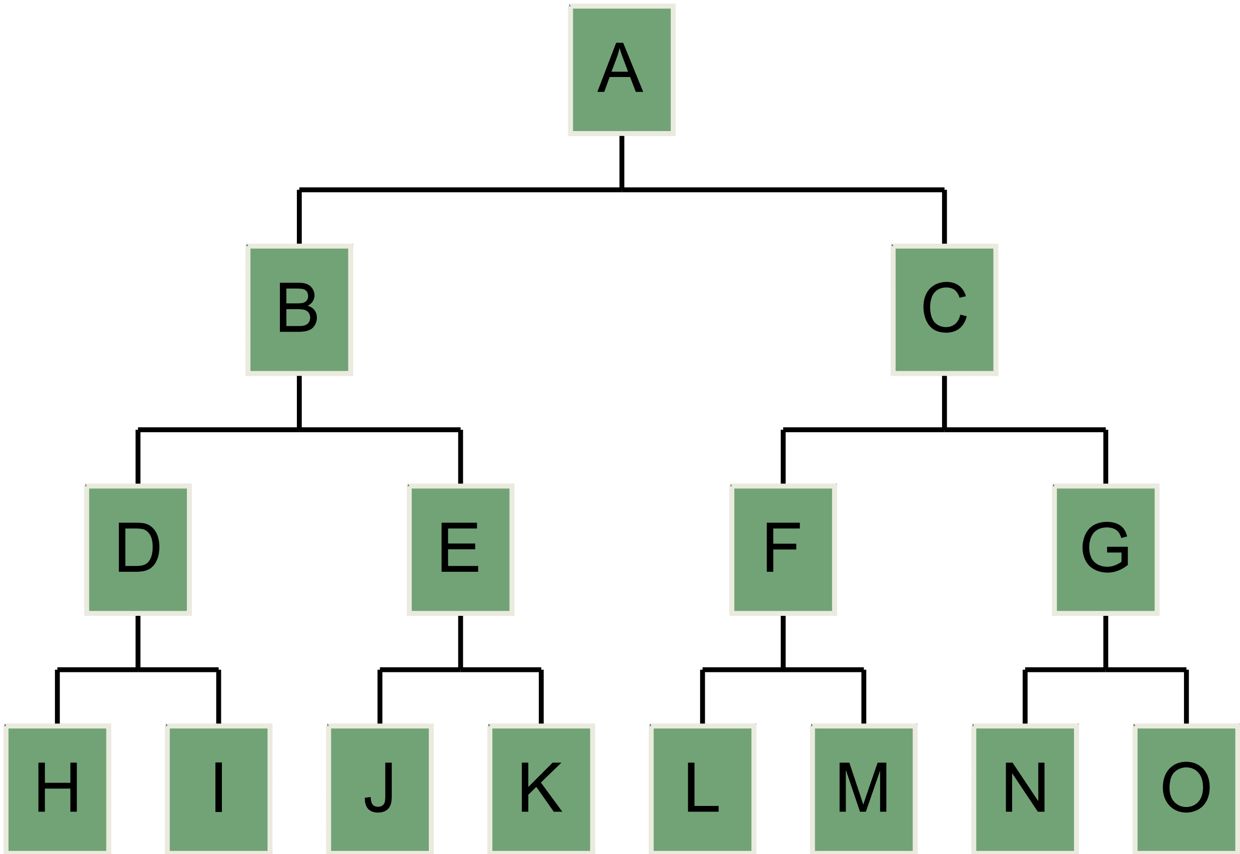
Árvore Binária

- árvore binária cheia de nível d: árvore binária estrita com todas as folhas no nível d

- árvore binária completa de nível d: uma árvore binária estrita com todas as folhas no nível d ou no nível d-1

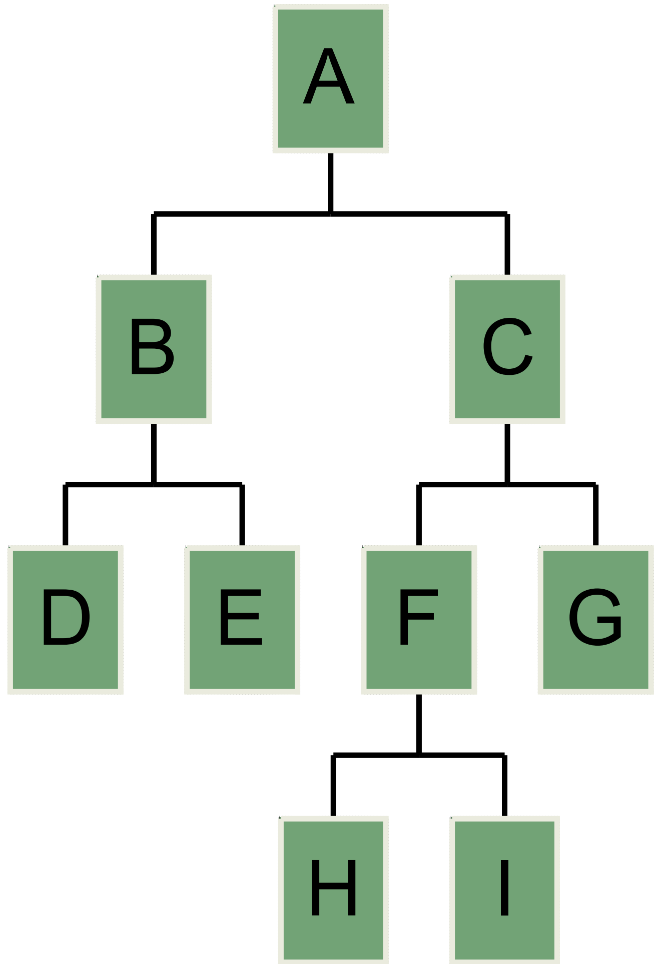
Árvore Binária Cheia

- árvore binária cheia de nível d:
  - árvore binária estrita com todas as folhas no nível d



Árvore Binária Completa

- árvore binária completa de nível d:
  - uma árvore binária estrita com todas as folhas no nível d ou no nível d-1

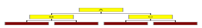


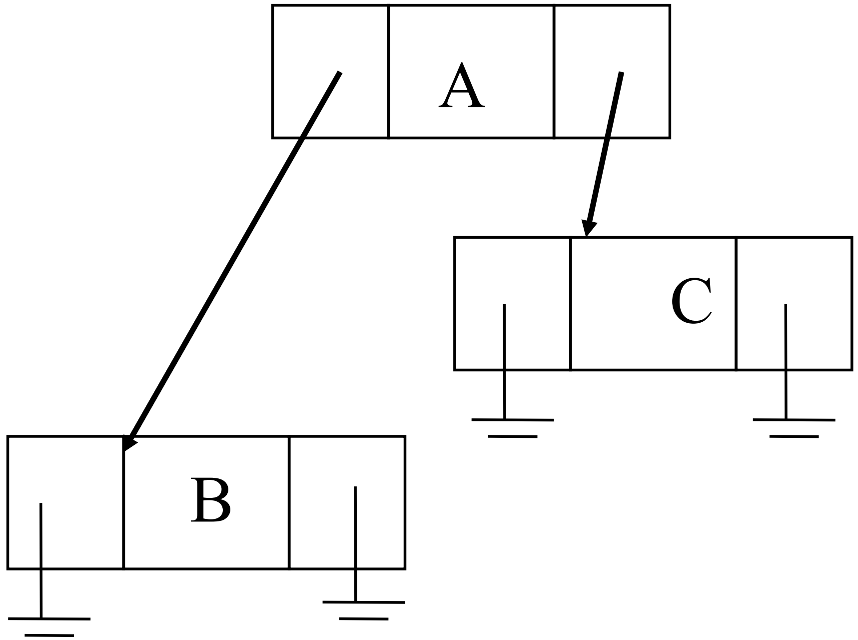
Árvore Binária

- Para muitas aplicações, é importante a relação entre altura e número de nós

Estrutura de dados do nó da AB

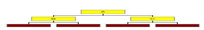

```
typedef struct {
        int info;
        tipo_no * esq;
        tipo_no * dir;
} tipo_no;
```

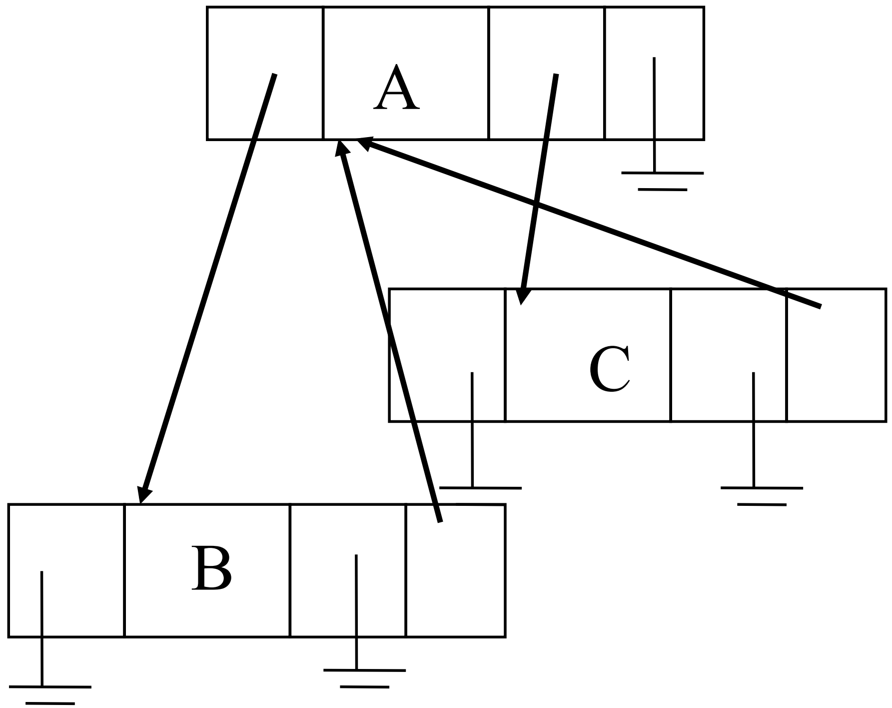




Estrutura de dados do nó da AB

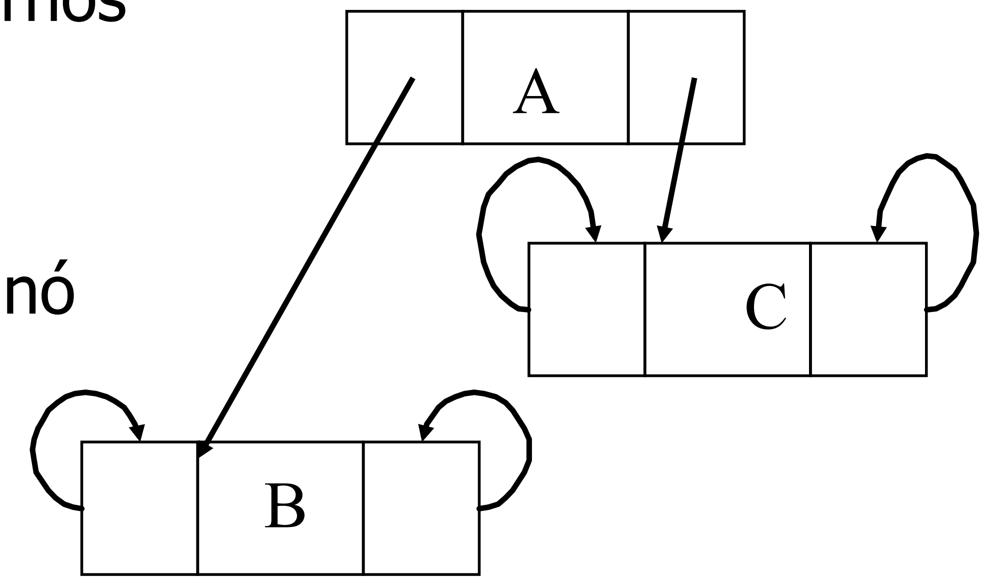

```
typedef struct {
        int info;
        tipo_no * esq;
        tipo_no * dir;
        tipo_no * pai;
} tipo_no;
```

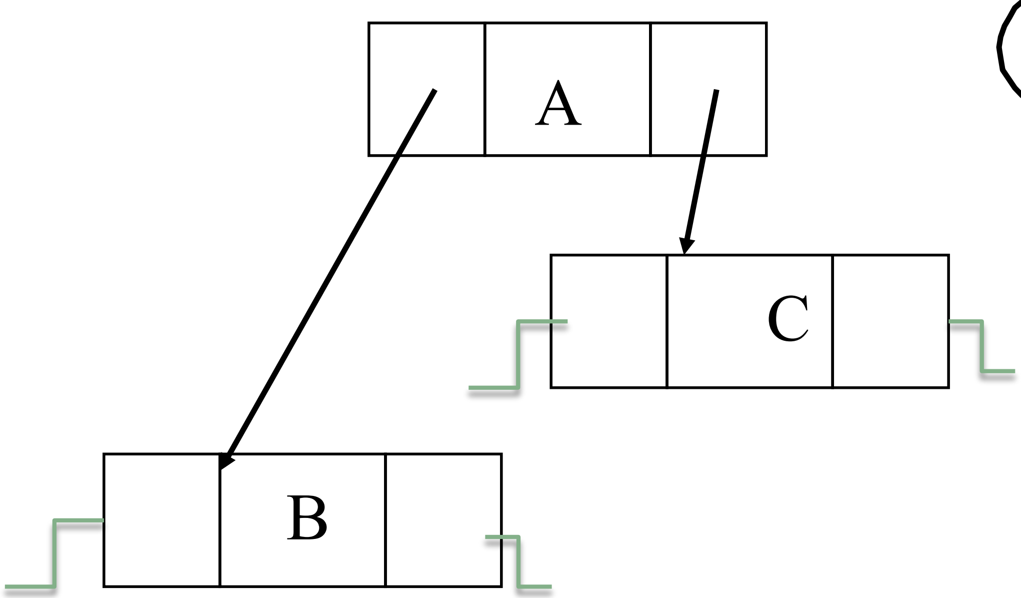




Representando Árvores Binárias

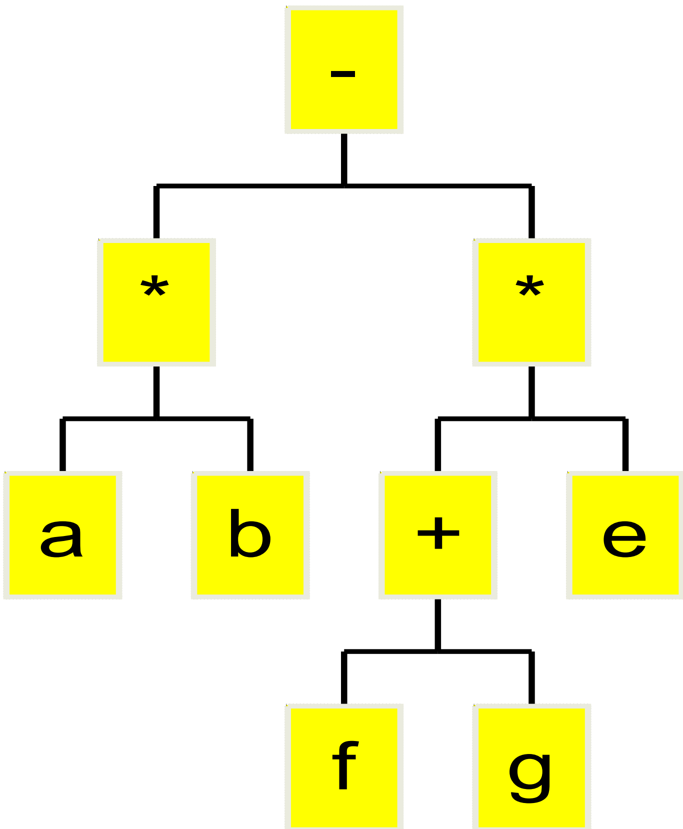
- Na representação nós externos podem ser
  - NULL
  - ponteiro para o próprio nó





Representando Árvores Binárias

a*b+d - e *(f+g)



Representando Florestas

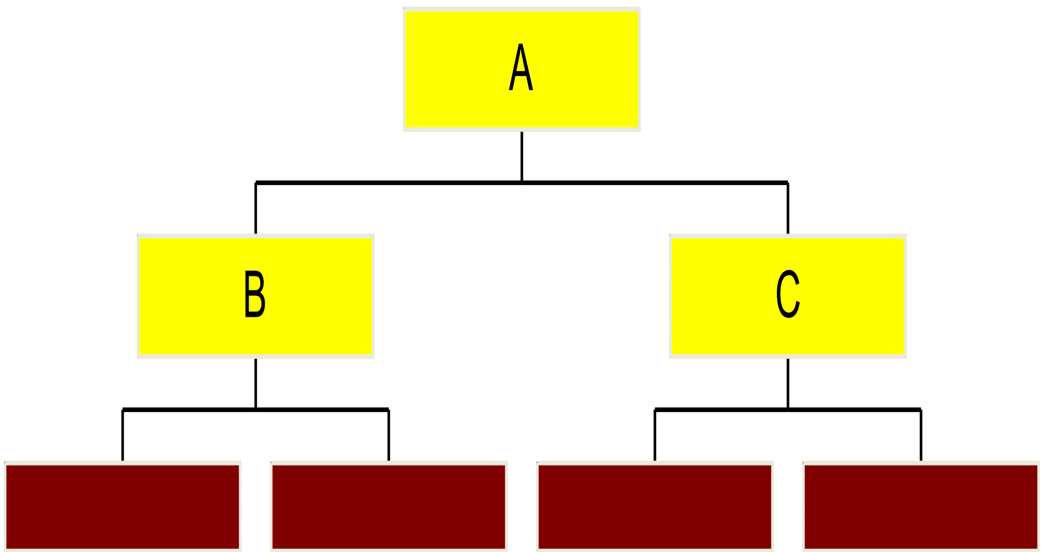
- árvores binárias possuem dois ponteiros em cada nó interno, um para cada filho, e portanto sua representação é imediata

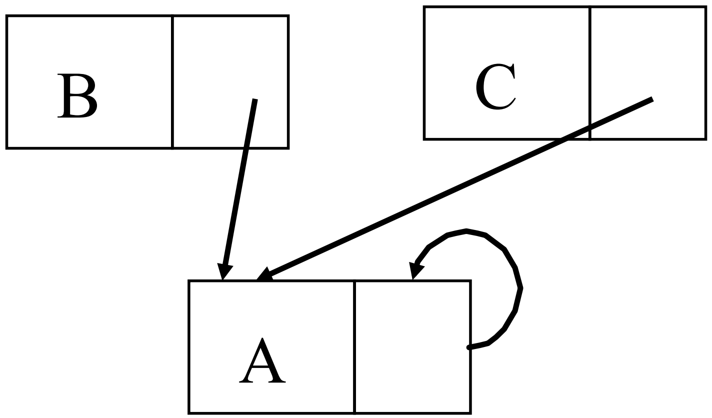
- o que fazer para árvores gerais ou florestas, com um número arbitrário de filhos por nó, que requerem um número arbitrário de ponteiros

Representação depende da Aplicação

Representando Florestas

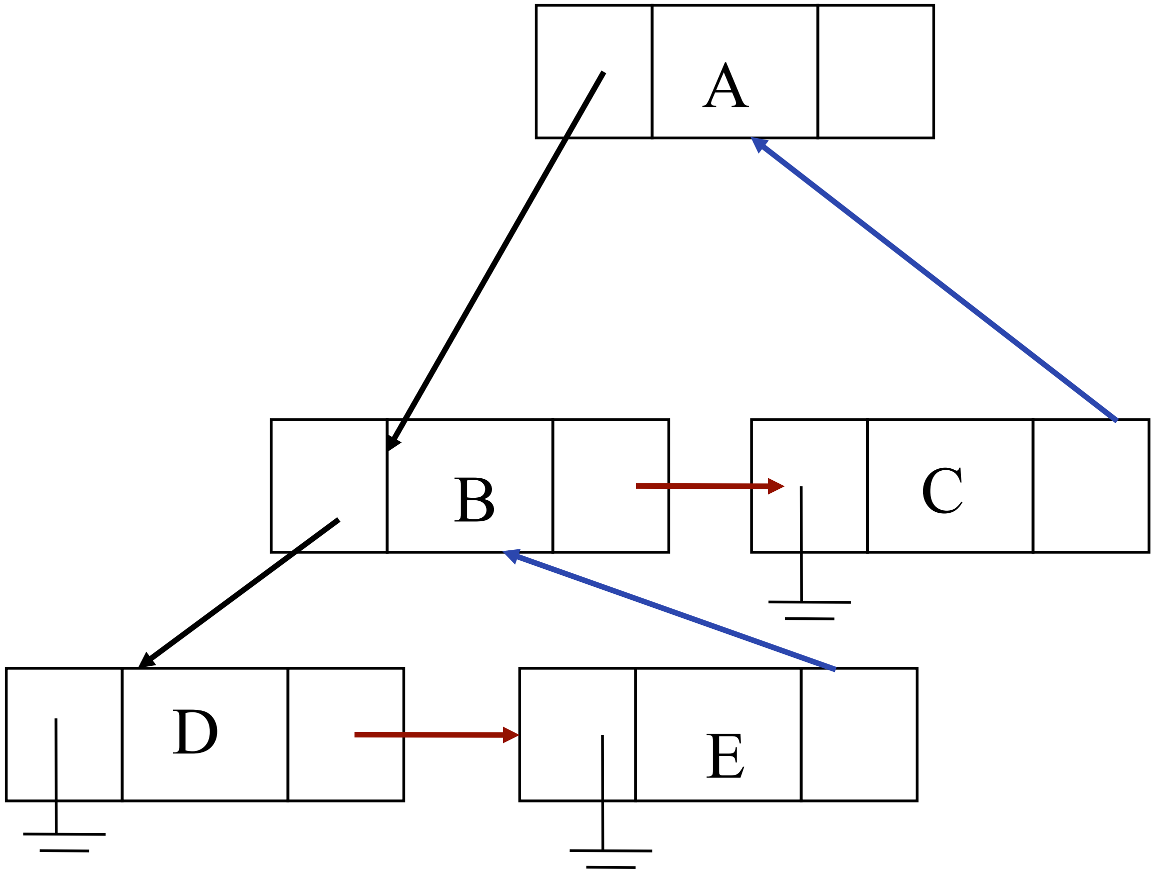
- se não é necessário caminhar para os níveis de baixo da árvore, mas só para os de cima
  - percorre-se a ávores dos nós terminais para os não terminais e por último a raiz





Representando Florestas

```
typedef struct {
      char info;
      tipo_no * filho;
      tipo_no * irmão;
} tipo_no
```



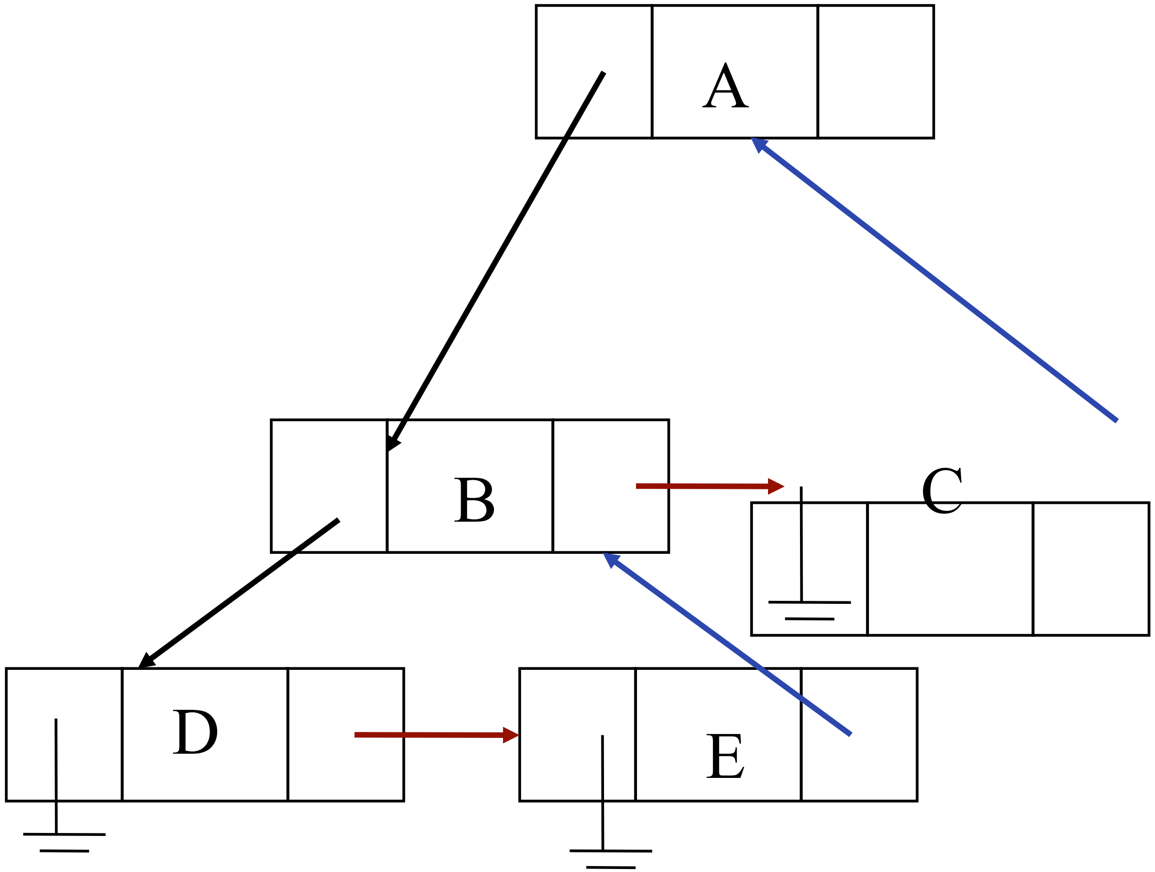
Representando Florestas

- se é necessário caminhar para os níveis mais altos
  - empregar uma lista encadeada conectando o nó com seus irmãos e outra com seus filhos

- ao invés de empregar um nó dummy para terminar cada lista pode-se apontar de volta para seu pai permitindo mover para cima ou para baixo
  - esses ponteiros para pai tem que estar marcados para poder ser distingui-los dos ponteiros para irmãos
  - alternativamente pode-se armazenar o nome do pai de forma que a busca pára quando o nó for revisitado

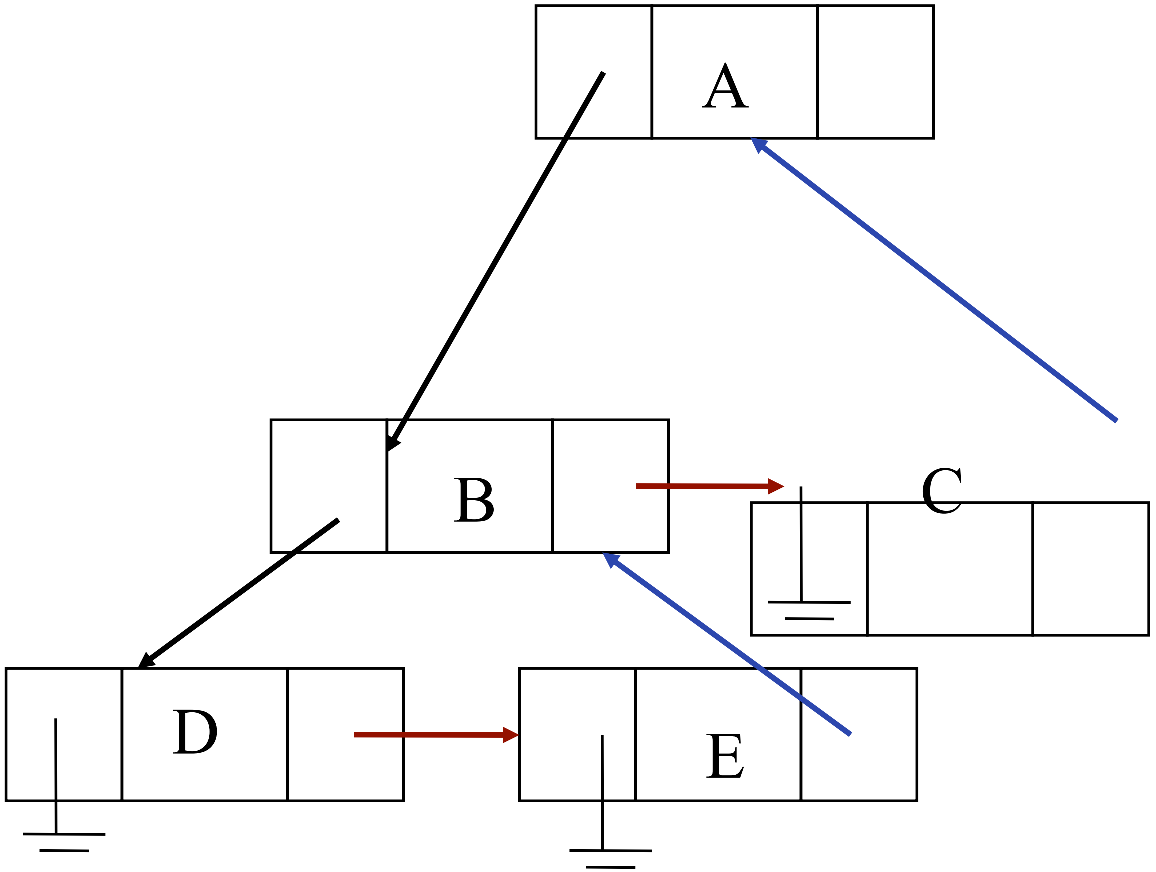
Representando Florestas


```
typedef struct {
      char info;
      tipo_no * filho;
      tipo_no * irmão;
} tipo_no
```



Representando Florestas


```
typedef struct {
      char info;
      tipo_no * filho;
      tipo_no * irmão;
} tipo_no
```



- então, lista encadeada unindo os irmãos
- o último ligado ao pai (pois não tem irmãos e nem filho)

Operações em Árvores Binárias

operação	significado
visita(p)	conteúdo do nó apontado por p
esquerda(p)	ponteiro para o filho da esquerda de p
direita(p)	ponteiro para o filho da direita de p
pai(p)	ponteiro para o pai de p
irmão(p)	ponteiro para o irmão de p
eh_esq(p)	Retorna true se p é filho da esquerda e false, se filho da direita
eh_dir(p)	Retorna true se p é filho da esquerda e false, se filho da direita

Operações em Árvores Binárias

operação	significado
p = cria(x)	Cria uma AB com apenas um nó com conteúdo x. Retorna o ponteiro para a nova árvore.
resp = filho_esq(p,x)	Cria um filho à esquerda do nó apontado por p, com conteúdo x. Retorna false caso já exista um filho esquerdo e true caso contário.
resp = filho_dir (p,x)	Cria um filho à direita do nó apontado por p, com conteúdo x. Retorna false caso já exista um filho direito e true caso contário.

Aplicações com Árvores Binárias

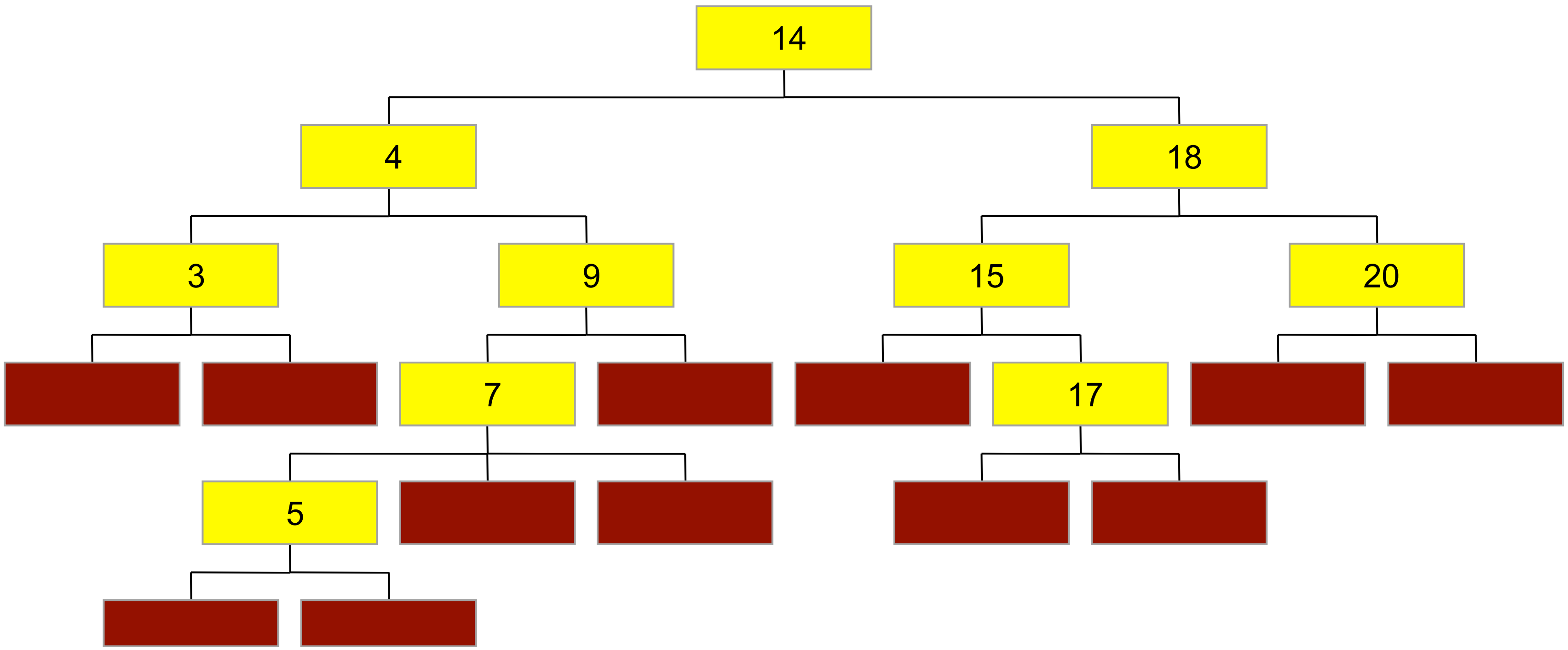
- É uma estrutura útil quando uma de duas decisões devem ser tomadas no decorrer do processo.
  - Encontrar todas as duplicatas em uma lista de números
  - Uma forma de fazer isso é comparar o número como todos os que o precedem
    - isto não é uma solução eficiente

Aplicações com Árvores Binárias

- Solução: empregar uma árvore binária
  - Armazenam-se os números na árvore de forma a:
    - o 1º número é armazenado na raiz de uma árvore com apenas um nó interno
    - cada um dos próximos números na lista é comparado com a raiz:
      - caso seja igual é uma duplicata
      - caso seja menor, é armazenado na sub-árvore da direita seguindo-se recursivamente o mesmo procedimento
      - caso seja maior, é armazenado na sub-árvore da esquerda seguindo-se recursivamente o mesmo procedimento

Aplicações com Árvores Binárias

14, 18, 4, 9, 7, 15, 3, 5, 17, 4, 20, 9, 5



Aplicações com Árvores Binárias

- outra aplicação comum é atravessar a árvore binária, visitando cada nó
  - como sistematicamente visitaremos cada nó?
- operação é trivial para listas lineares
- para árvores, existem diferentes formas de proceder
  - os métodos diferem conforme a ordem em que se visitam os nós, o problema sendo resolvido

Atravessando Árvores Binárias

- Métodos
  - **pré-ordem**:visite a raiz, então visite a subárvore da esquerda, depois a subárvore da direita
  - **em-ordem ou ordem simétrica:** visite a subárvore da esquerda, então visite a raiz, depois a subárvore da direita
  - **pós-ordem**: visite a subárvore da esquerda, então visite a subárvore da direita, depois a raiz

Atravessando Árvores Binárias

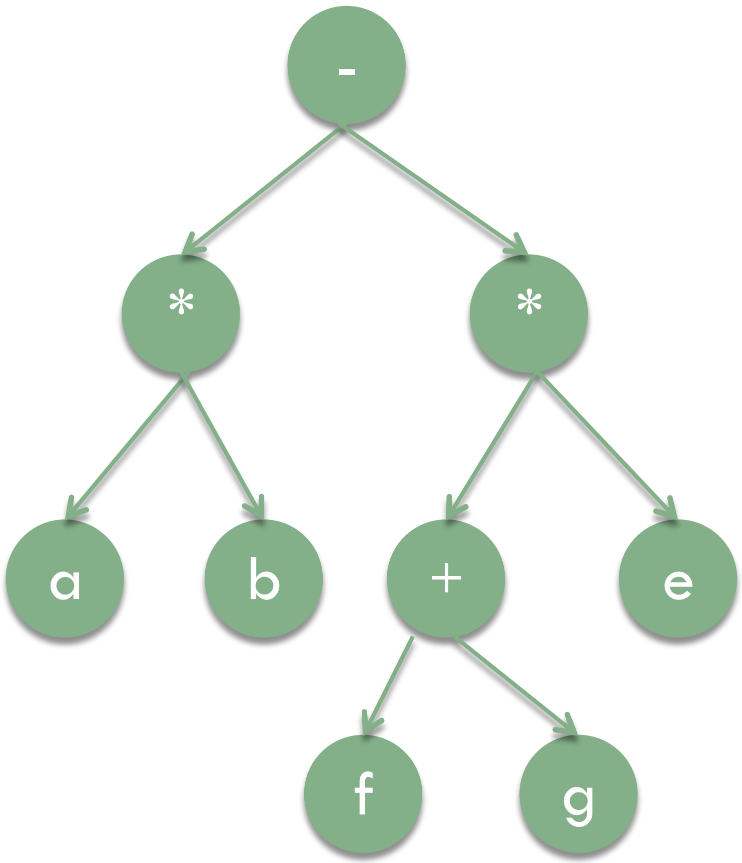
- implementação simples dos métodos - recursiva
  - como se visita uma subárvore de cada vez, seguindo-se a regra recursiva , cada subárvore é visitada começando pela raiz
  - no procedimento principal, começa visitando a raiz

Pré-ordem

```
pre_ordem (pt)
{
    if (pt == NULL) return ();
    visite(pt);
    pre_ordem (pt->esq);
    pre_ordem (pt-> dir);
}
```

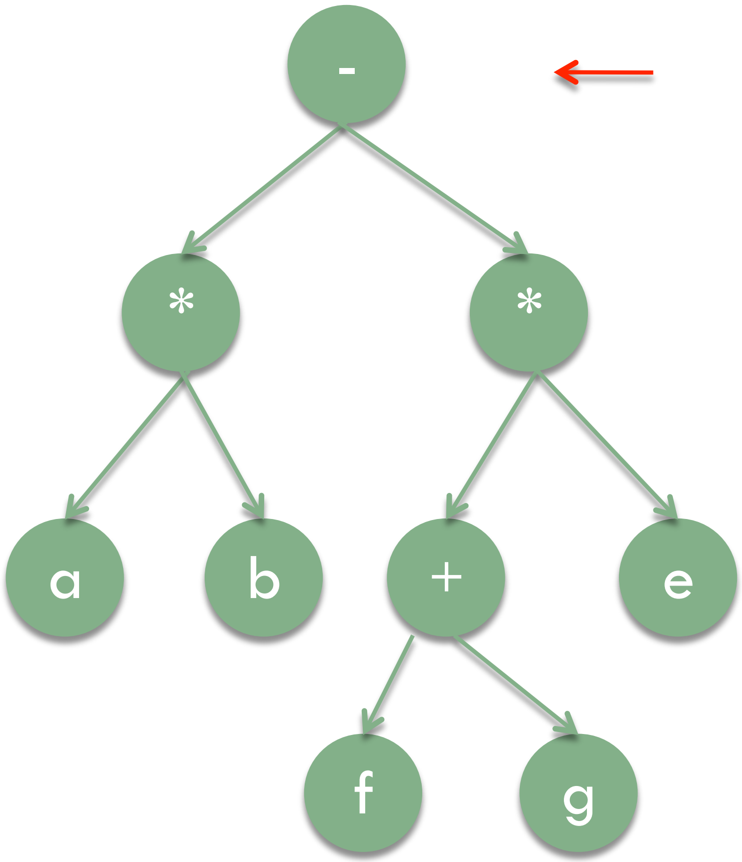
Atravessando Árvores Binárias

$a*b - (f + g) * e$



Atravessando Árvores Binárias

a*b – (f + g) * e

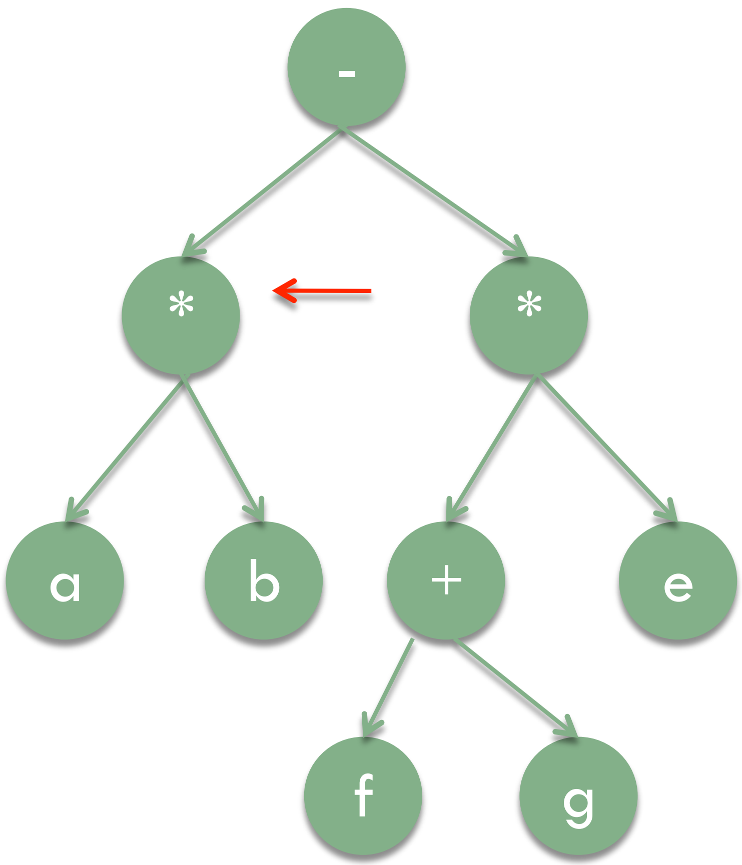


pré-ordem:

-

Atravessando Árvores Binárias

a*b – (f + g) * e

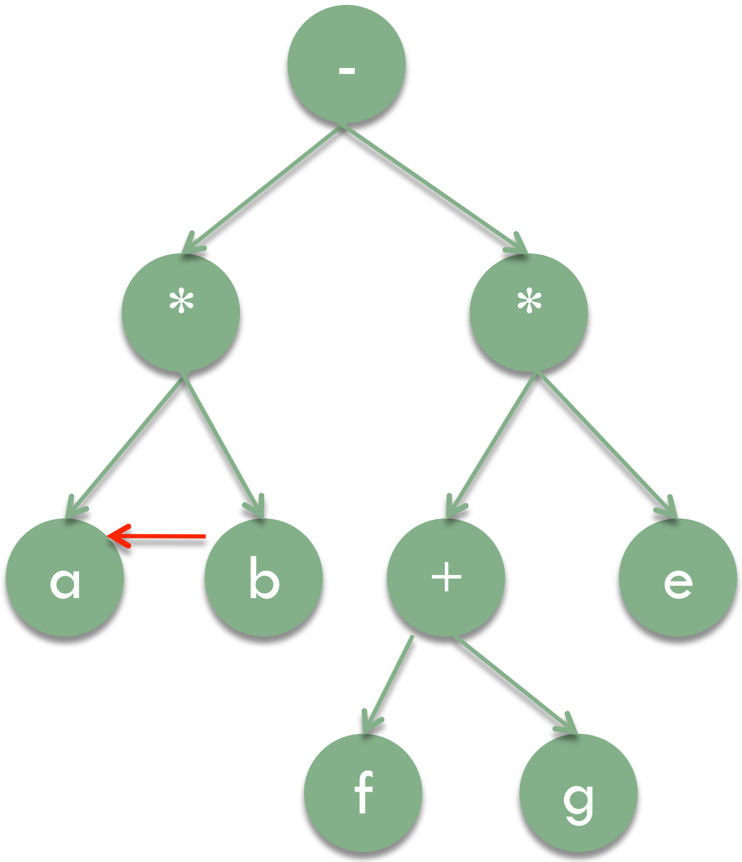


pré-ordem:

- *

Atravessando Árvores Binárias

a*b – (f + g) * e

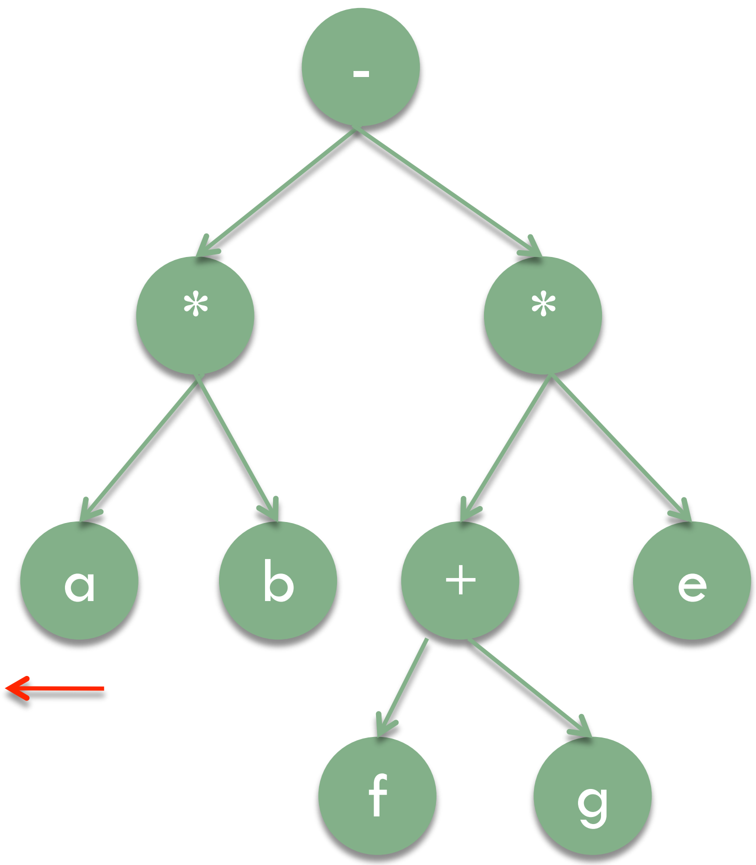


pré-ordem:

- * a

Atravessando Árvores Binárias

a*b – (f + g) * e

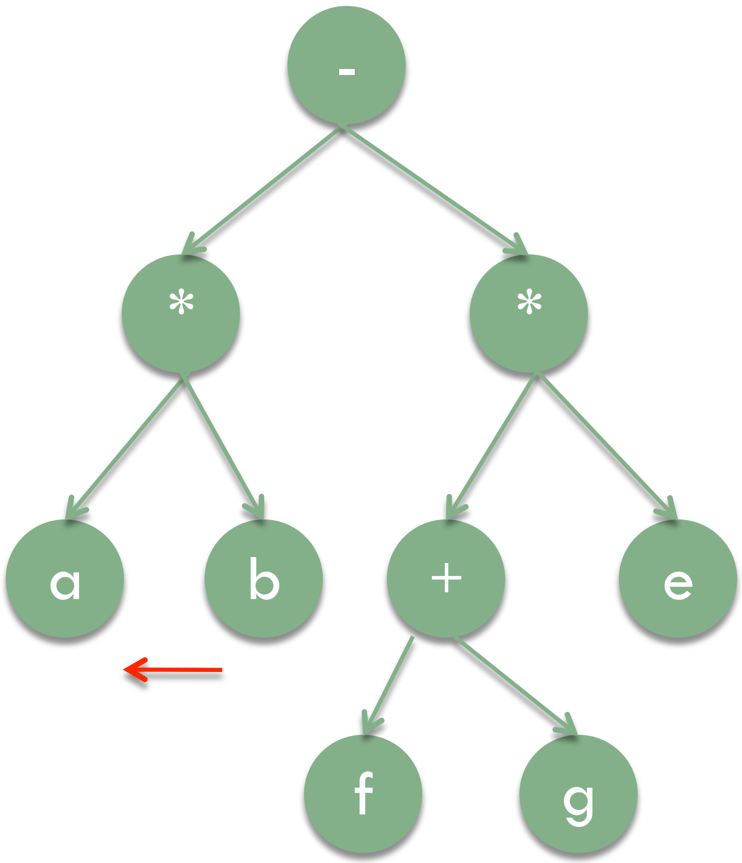


pré-ordem:

\- * a

Atravessando Árvores Binárias

a*b – (f + g) * e

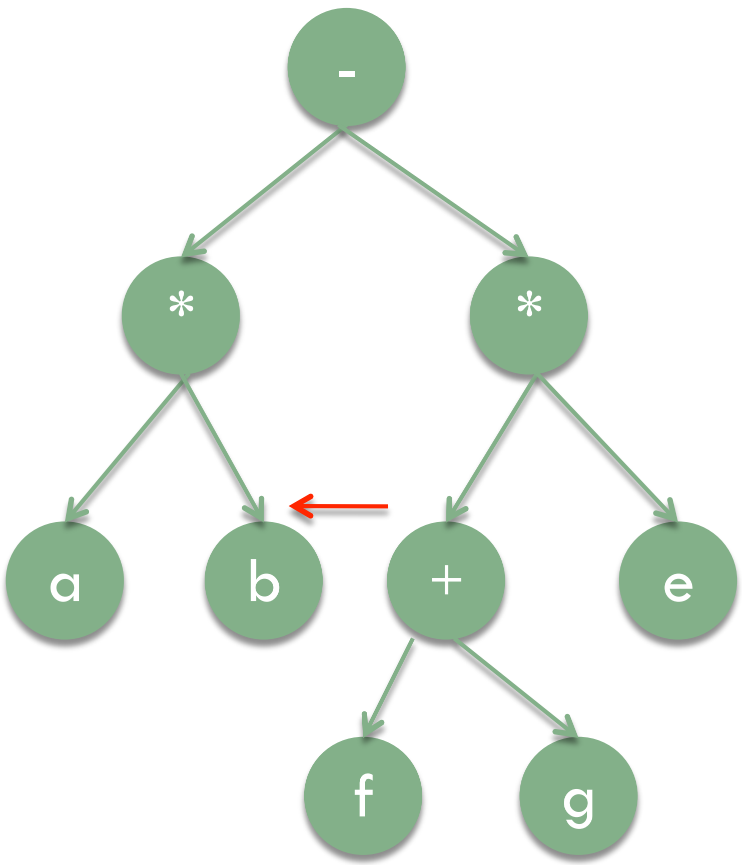


pré-ordem:

- * a

Atravessando Árvores Binárias

a*b – (f + g) * e

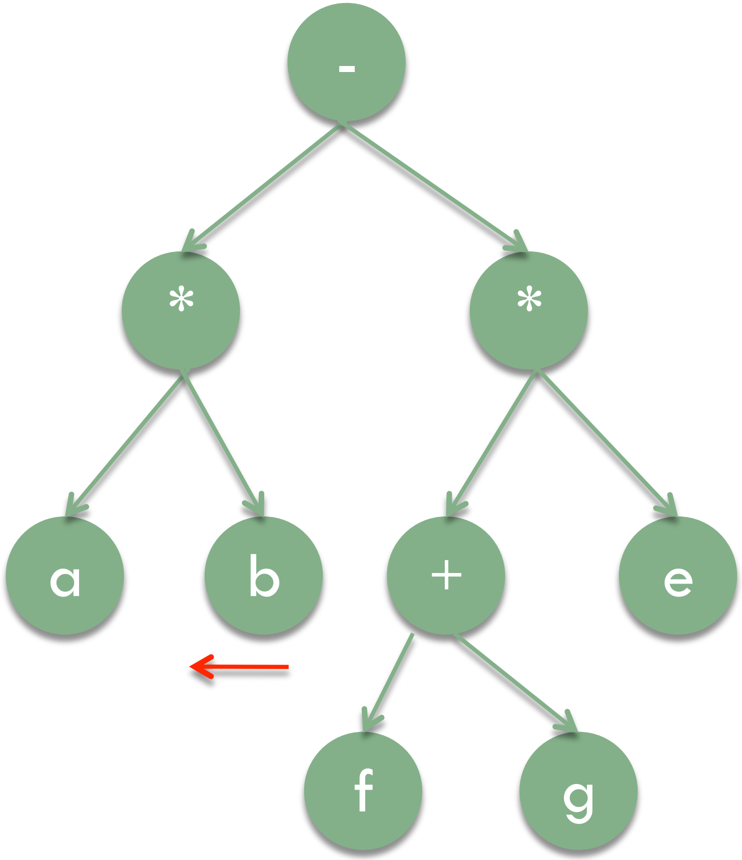


pré-ordem:

- * a b

Atravessando Árvores Binárias

a*b – (f + g) * e

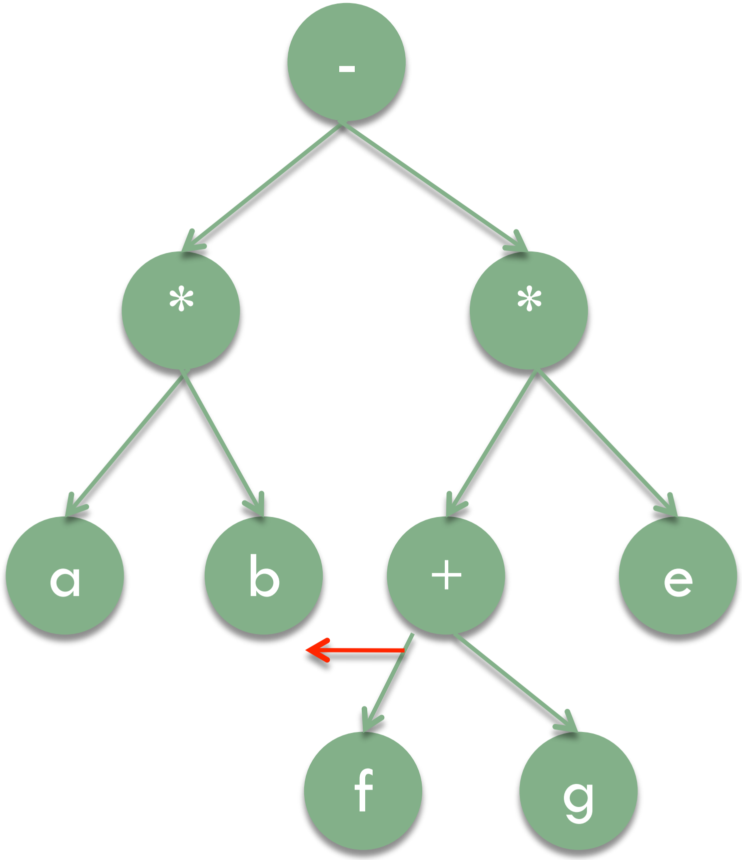


pré-ordem:

- * a b

Atravessando Árvores Binárias

a*b – (f + g) * e

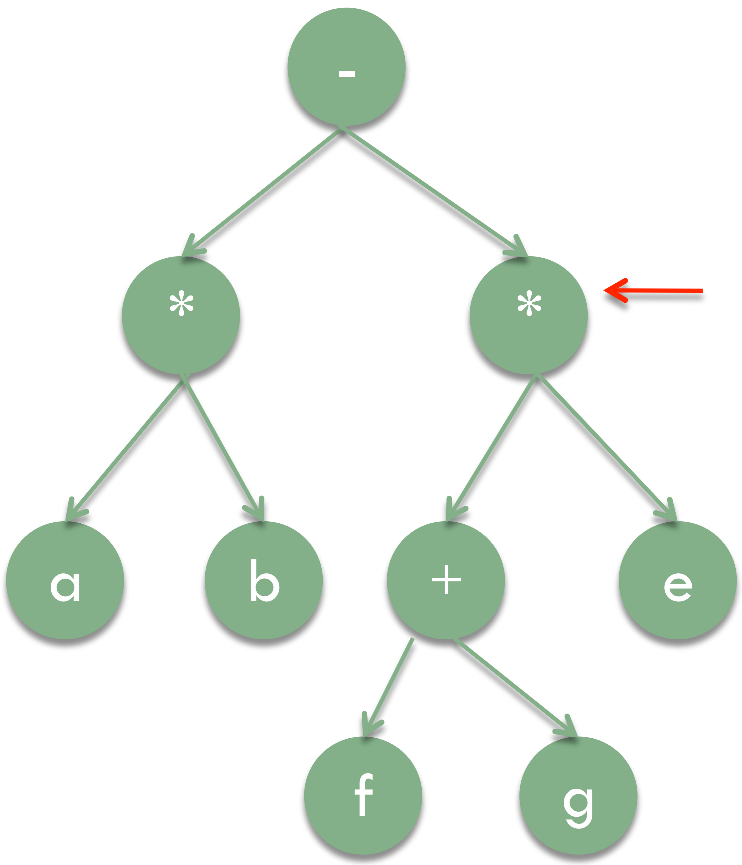


pré-ordem:

- * a b

Atravessando Árvores Binárias

a*b – (f + g) * e

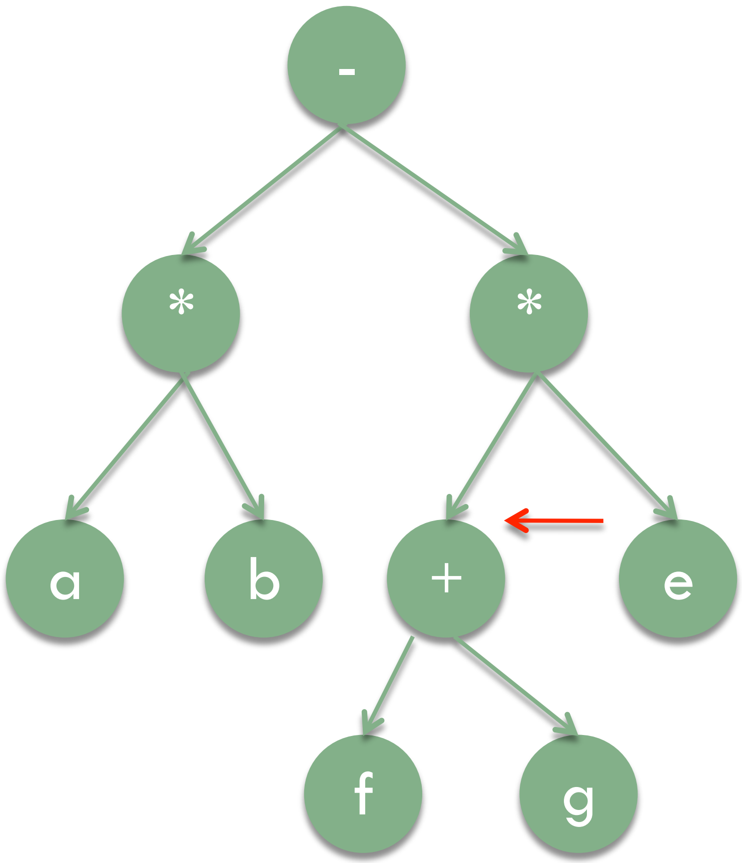


pré-ordem:

- * a b *

Atravessando Árvores Binárias

a*b – (f + g) * e

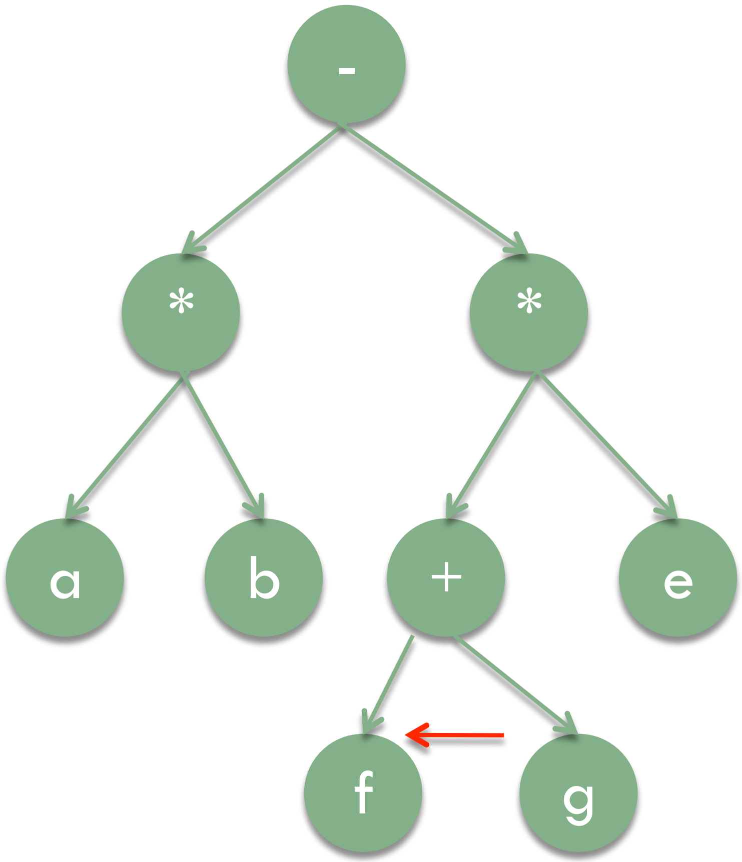


pré-ordem:

\- * a b *+

Atravessando Árvores Binárias

a*b – (f + g) * e

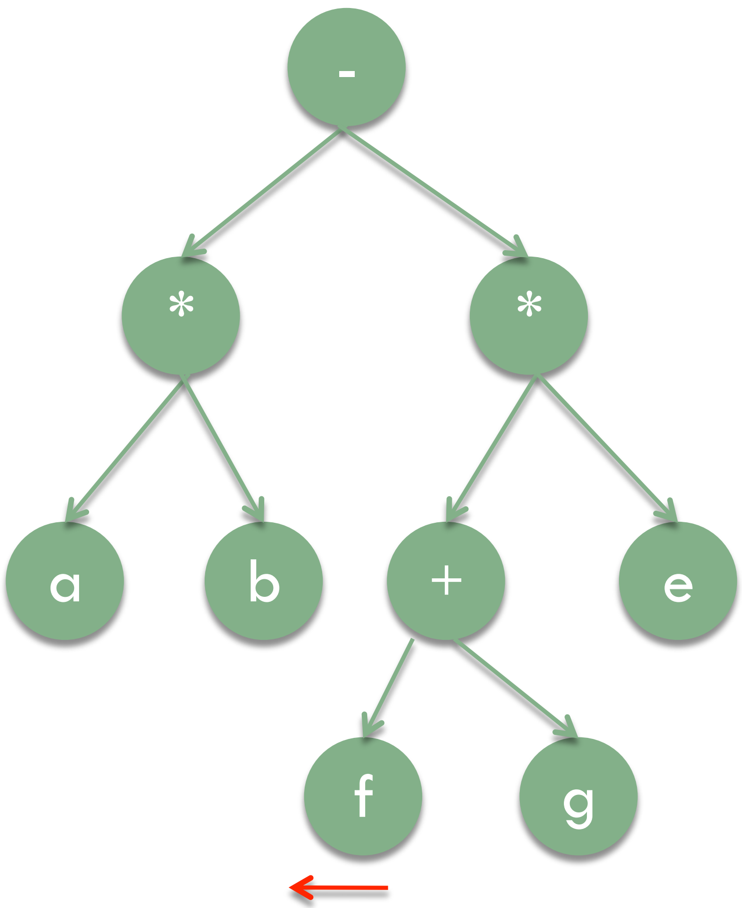


pré-ordem:

- * a b *+f

Atravessando Árvores Binárias

a*b – (f + g) * e

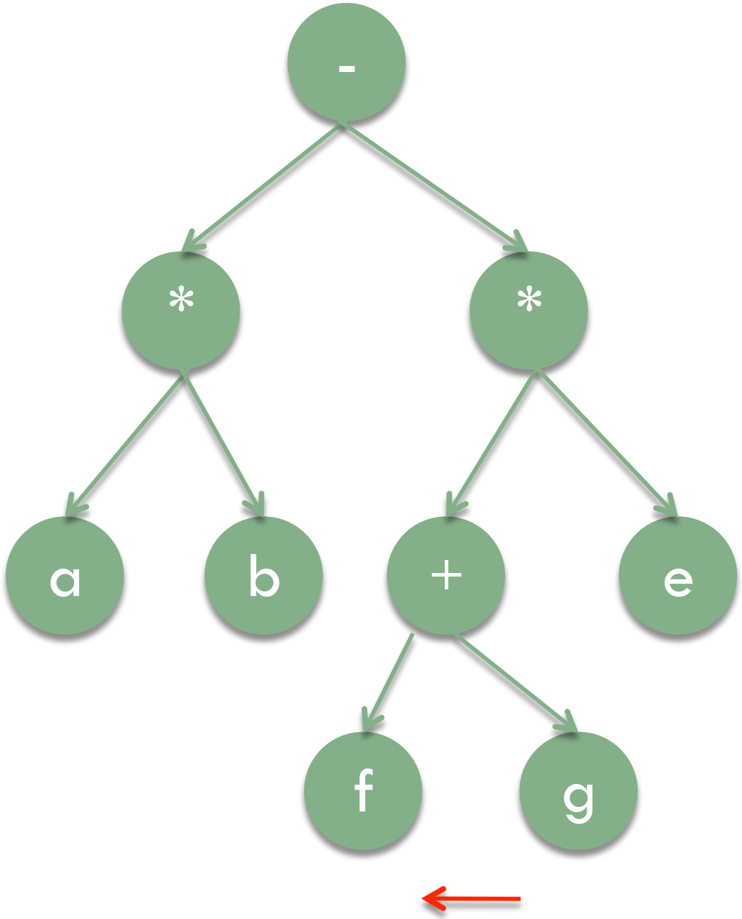


pré-ordem:

- * a b *+f

Atravessando Árvores Binárias

a*b – (f + g) * e

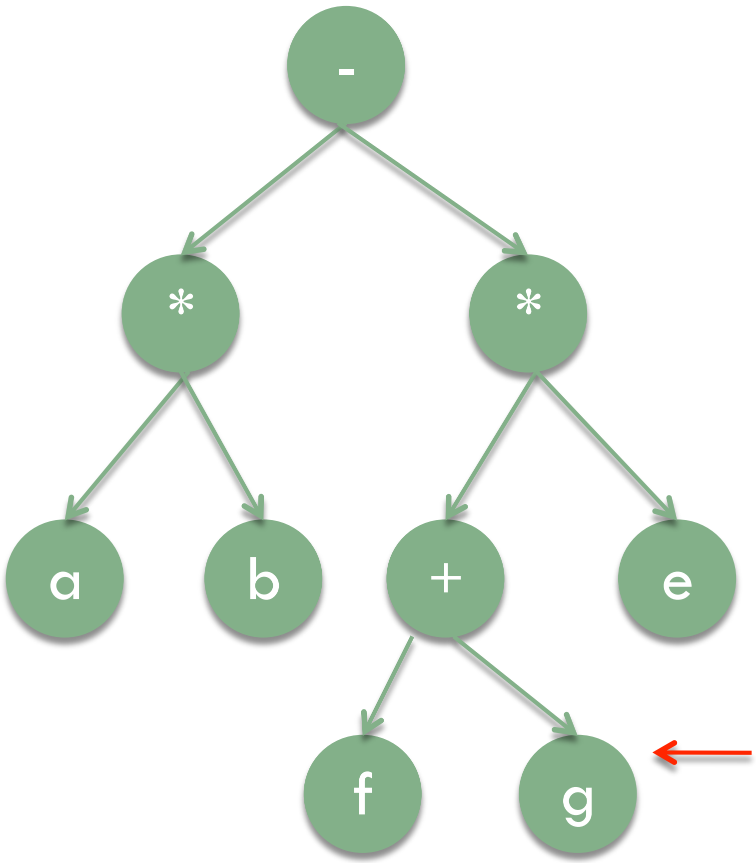


pré-ordem:

- * a b *+f

Atravessando Árvores Binárias

a*b – (f + g) * e

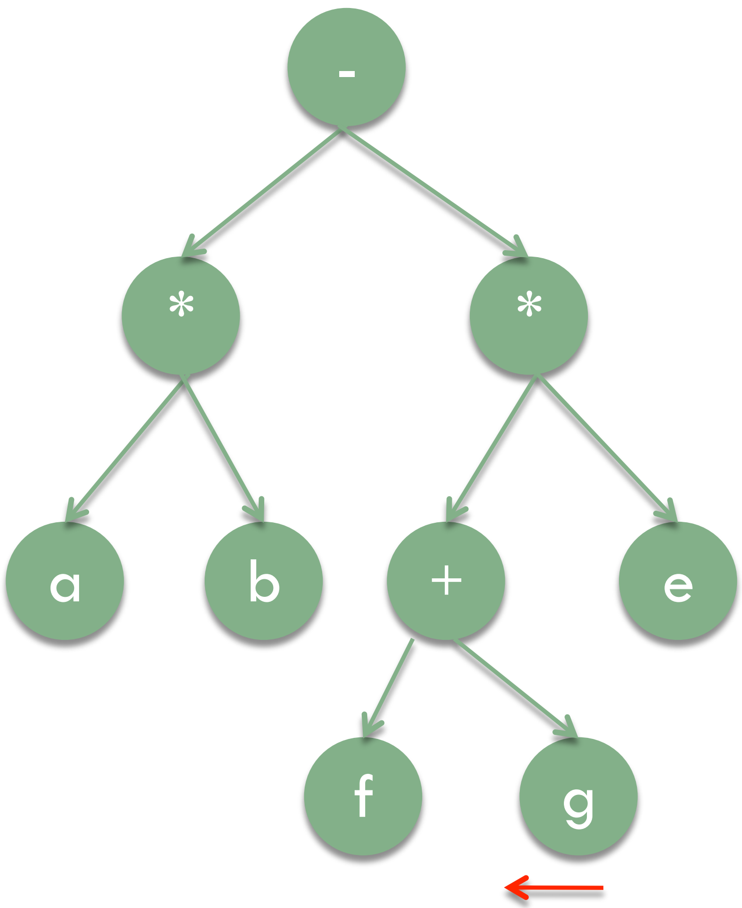


pré-ordem:

- * a b *+f g

Atravessando Árvores Binárias

a*b – (f + g) * e

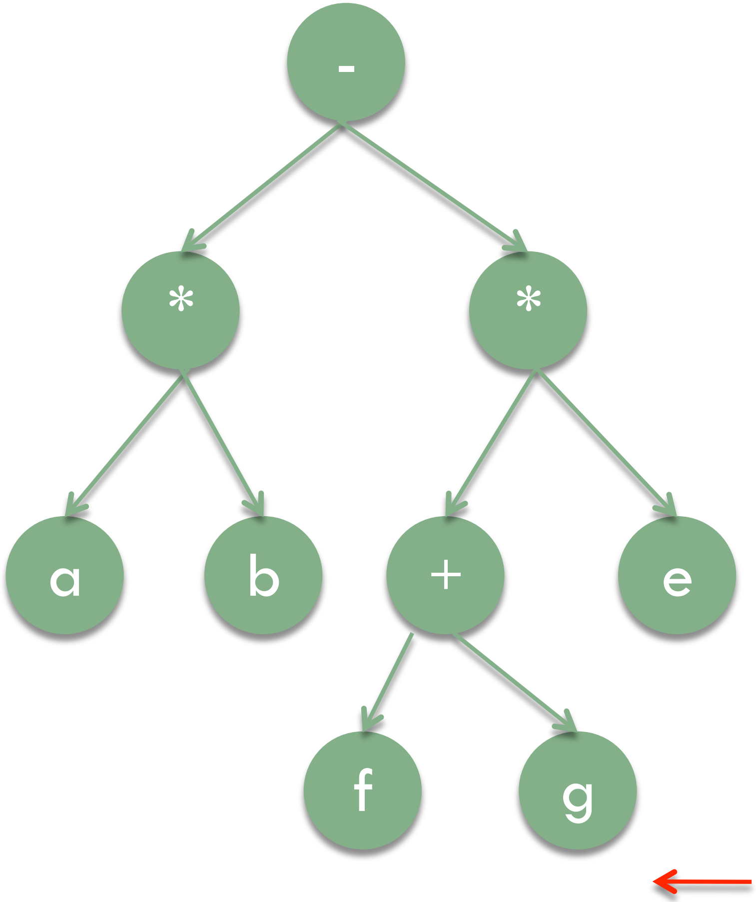


pré-ordem:

- * a b *+f g

Atravessando Árvores Binárias

a*b – (f + g) * e

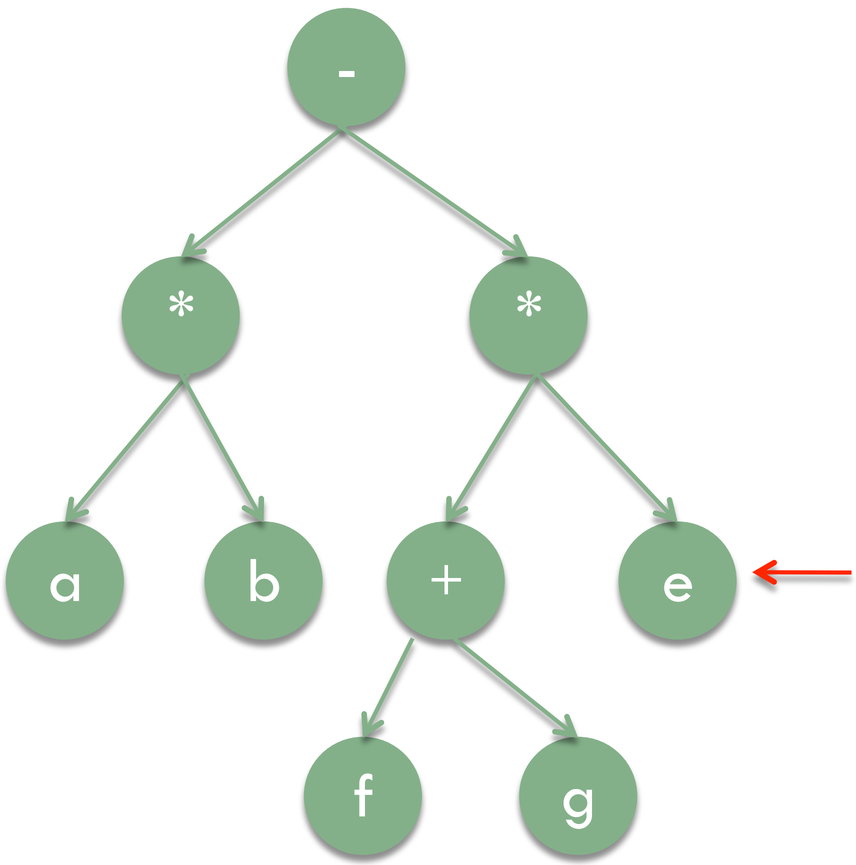


pré-ordem:

- * a b *+f g

Atravessando Árvores Binárias

$a*b - (f + g) * e$

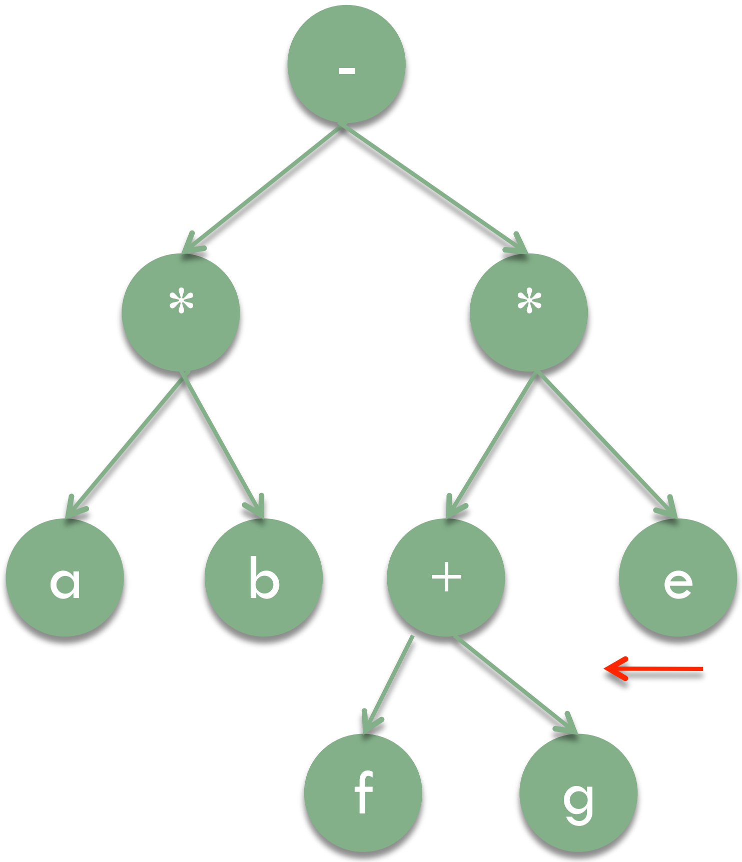


pré-ordem:

- * a b *+f g e

Atravessando Árvores Binárias

$a*b - (f + g) * e$

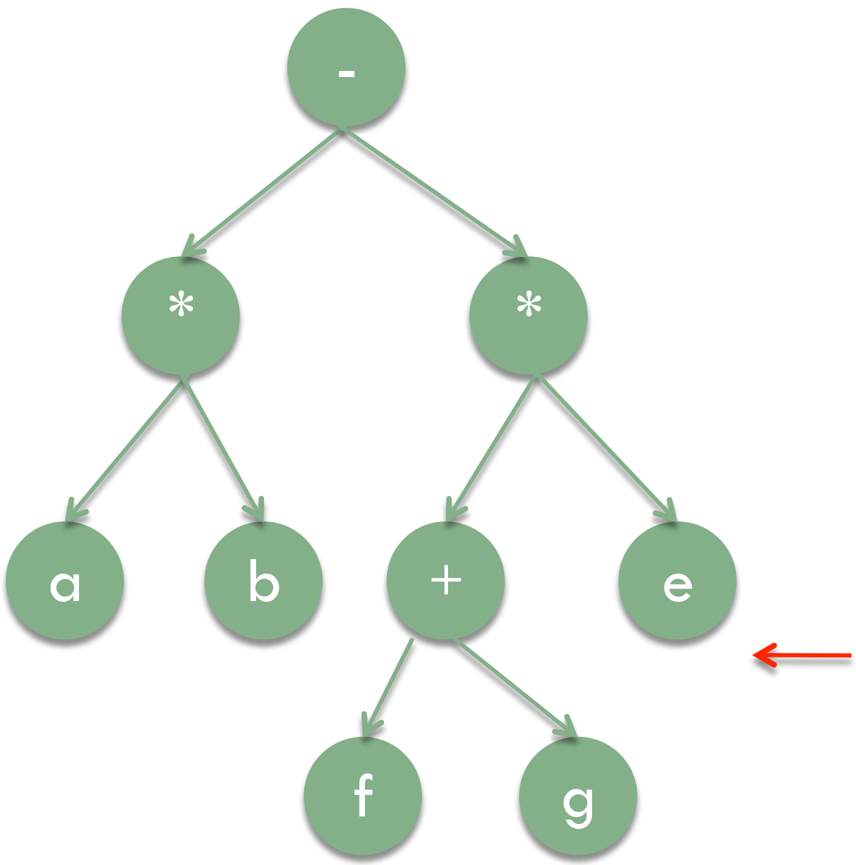


pré-ordem:

- * a b *+f g e

Atravessando Árvores Binárias

a*b – (f + g) * e



pré-ordem:

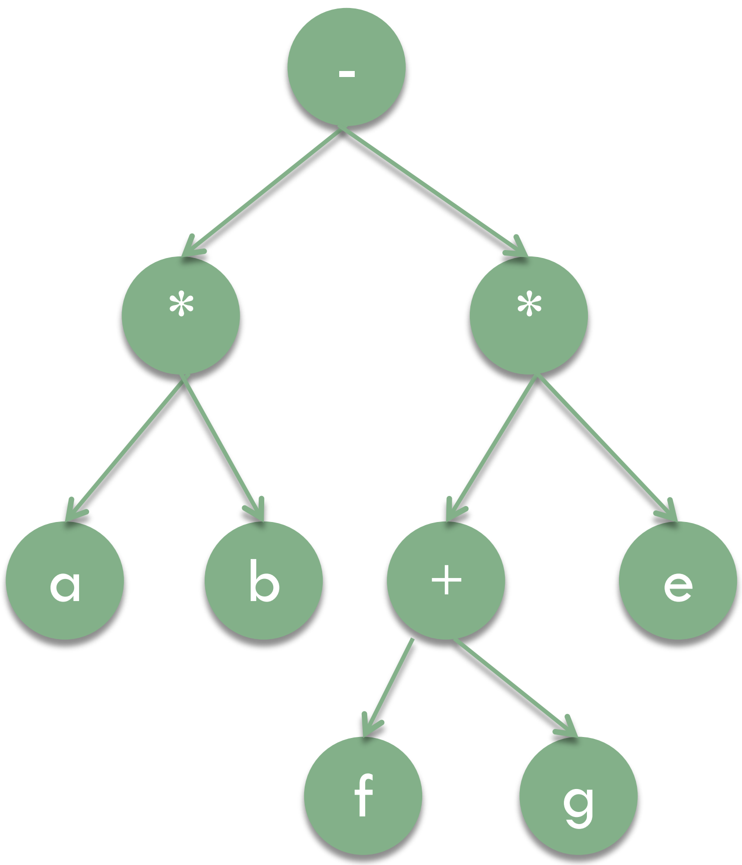
-* a b *+f g e

em-ordem

```
em_ordem (pt)
{
    if (pt == NULL) return ();
    em_ordem (pt->esq);
    visite(pt);
    em_ordem (pt-> dir);
}
```

Atravessando Árvores Binárias

a*b – (f + g) * e



em-ordem:

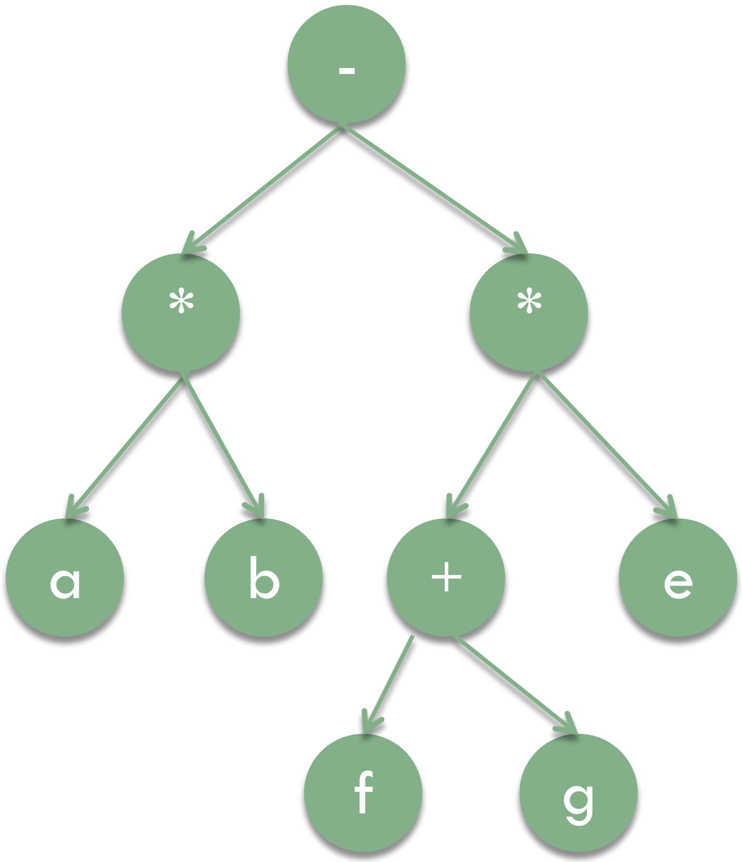
-a*b - f+g * e

Pós-ordem

```
pos_ordem (pt)
{
    if (pt == NULL) return ();
    pos_ordem (pt->esq);
    pos_ordem (pt-> dir);
    visite(pt);
}
```

Atravessando Árvores Binárias

a*b – (f + g) * e



pós-ordem:

a b * f g + e * -

Exercícios

1) Calcular a altura de cada nó de uma árvore binária

2) Implementar os procedimentos em-ordem de forma não recursiva