Árvores Binárias II

05/11/2025

Ficheiro ZIP

- Está disponível no Moodle um ficheiro ZIP de suporte aos tópicos de hoje
- Atualização do tipo abstrato Árvore Binária de Inteiros
- O tipo abstrato Árvore Binária de Procura
- Funções incompletas, que permitem trabalho autónomo de desenvolvimento e teste

Sumário

- Recap
- Representação de expressões algébricas
- Travessias recursivas em Pré-Ordem, Em-Ordem e Pós-Ordem
- Travessia iterativa, por níveis, usando uma FILA / QUEUE
- Travessia iterativa, em pré-ordem, usando uma PILHA / STACK
- O TAD Árvore Binária de Procura – BST : Binary Search Tree
- Exercícios / Tarefas 

Recapitulação

TAD **Árvore Binária** – Funcionalidades

- Conjunto de elementos do mesmo tipo
- Armazenados **sem qualquer ordem** particular

- Procura / inserção / remoção / substituição
- Pertença

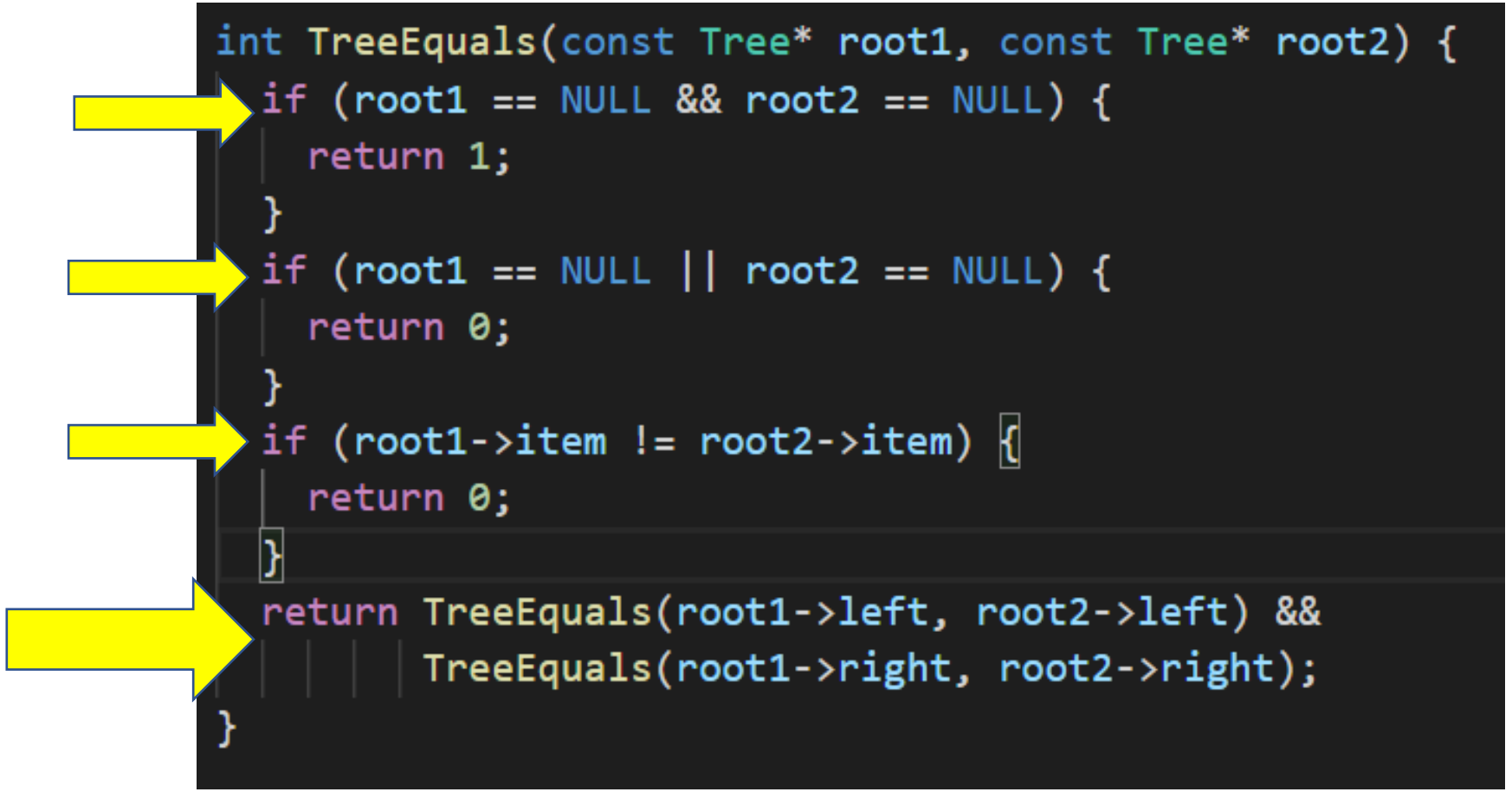
- search() / insert() / remove() / replace()
- contains()
- size() / isEmpty()
- create() / destroy()

Determinar a altura de uma árvore binária

```
int TreeGetHeight(const Tree* root) {
  if (root == NULL) return -1;

  int heightLeftSubTree = TreeGetHeight(root->left);

  int heightRightSubTree = TreeGetHeight(root->right);

  if (heightLeftSubTree > heightRightSubTree) {
    return 1 + heightLeftSubTree;
  }

  return 1 + heightRightSubTree;
}
```

- Árvore vazia tem altura -1
- Número de arcos da raiz até à folha mais longínqua

Verificar se duas árvores são iguais



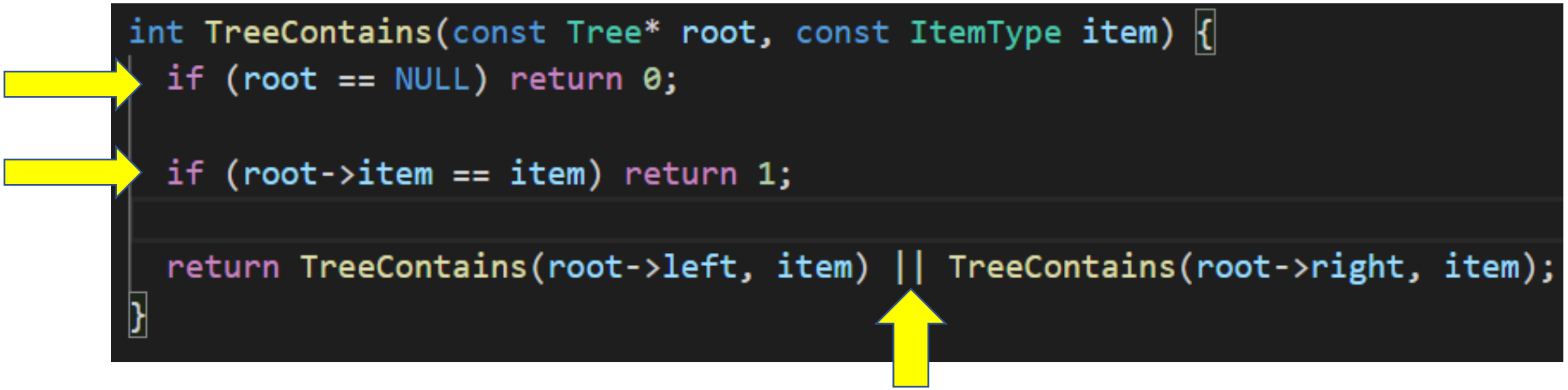
```
int TreeEquals(const Tree* root1, const Tree* root2) {
    if (root1 == NULL && root2 == NULL) {
        return 1;
    }
    if (root1 == NULL || root2 == NULL) {
        return 0;
    }
    if (root1->item != root2->item) {
        return 0;
    }
    return TreeEquals(root1->left, root2->left) &&
           TreeEquals(root1->right, root2->right);
}
```

- Casos de base
- Comparar as subárvores

Um item pertence à árvore ? – **Fizeram ?**

- Qual é a **estratégia recursiva** ?

Um item pertence à árvore ?



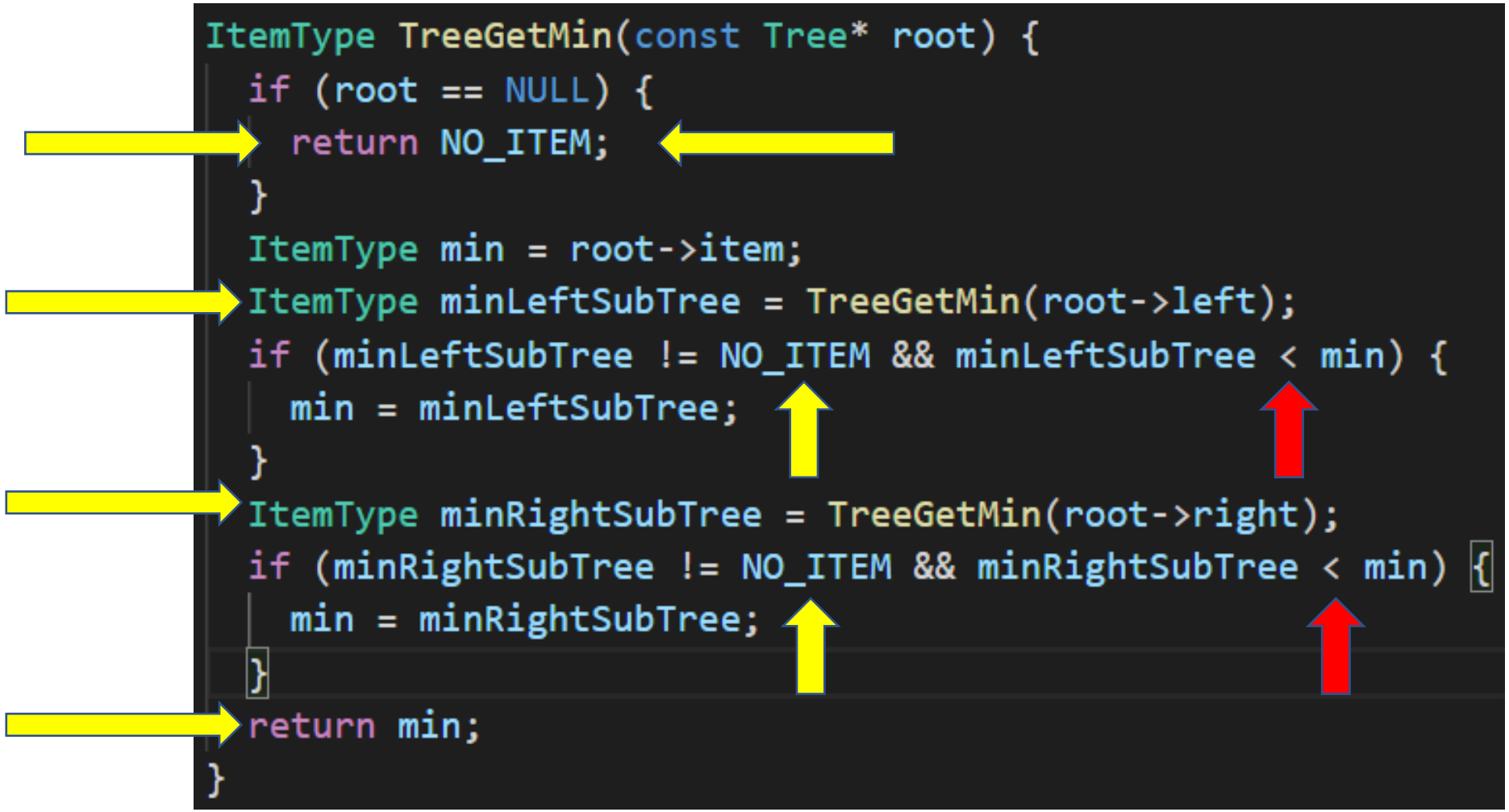
```
int TreeContains(const Tree* root, const ItemType item) {
  if (root == NULL) return 0;

  if (root->item == item) return 1;

  return TreeContains(root->left, item) || TreeContains(root->right, item);
}
```

- Eficiência ?
  - Pode ser necessário visitar **todos os nós** !
  - **Como evitar ?** – Associar uma **ordem** aos **itens** armazenados – **Como** ?

Qual é o menor elemento ? – **Fizeram ?**

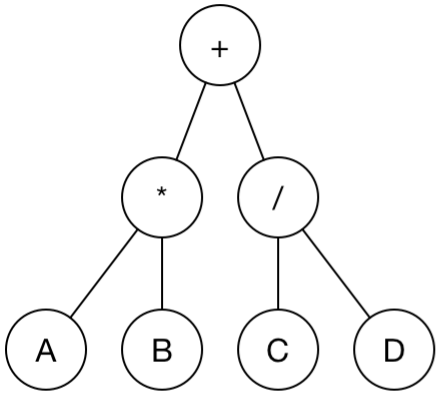
- Qual é a **estratégia recursiva** ?

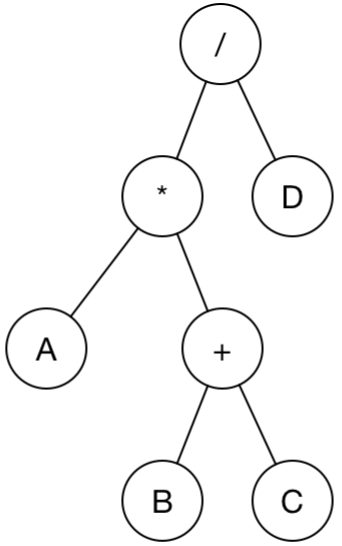
Qual é o menor elemento ? – Eficiência ?

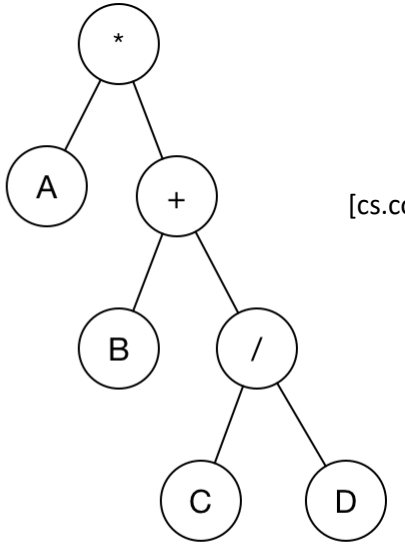


```
ItemType TreeGetMin(const Tree* root) {
  if (root == NULL) {
    return NO_ITEM;
  }
  ItemType min = root->item;
  ItemType minLeftSubTree = TreeGetMin(root->left);
  if (minLeftSubTree != NO_ITEM && minLeftSubTree < min) {
    min = minLeftSubTree;
  }
  ItemType minRightSubTree = TreeGetMin(root->right);
  if (minRightSubTree != NO_ITEM && minRightSubTree < min) {
    min = minRightSubTree;
  }
  return min;
}
```

- Árvore vazia
- Encontrar o menor de cada uma das subárvores
- Comparar entre si e com o valor da raiz







[cs.colostate.edu]

```
((A * B) + (C / D))
```

```
((A * (B + C)) / D)
```

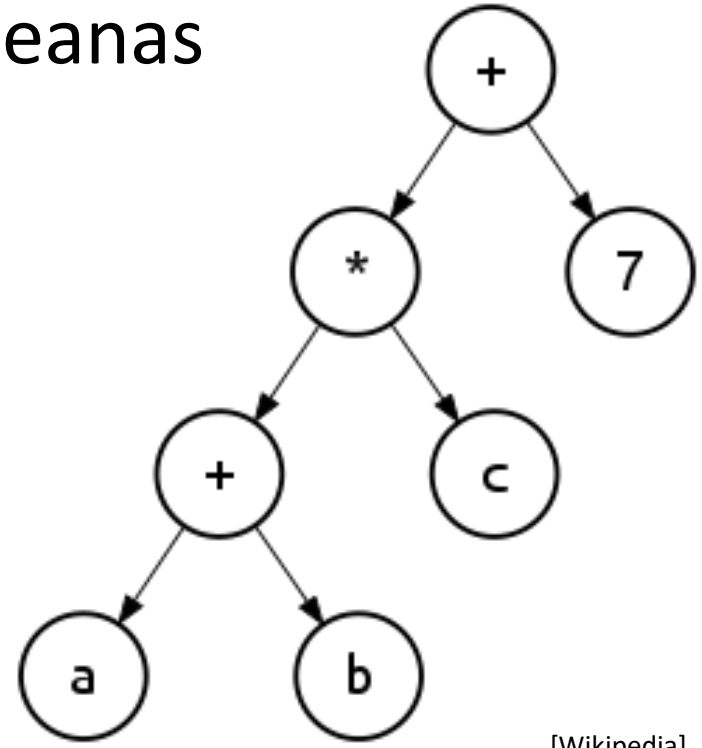
```
(A * (B + (C / D)))
```

Representação de expressões

Representação usando uma árvore binária

- Expressões aritméticas / algébricas / booleanas
- Folha : operando
- Nó não terminal : operador
- Não são necessários parênteses !!

- Expressão ? Notações possíveis ?
- Que travessias são possíveis ?



[Wikipedia]

Notação – Como representar uma expressão?

- Notação INFIXA : operando operador operando
- Notação PREFIXA : operador operando operando
- Notação POSFIXA : operando operando operador

PREFIX	POSTFIX	INFIX
* + a b c	a b + c *	(a + b) * c
+ a * b c	a b c * +	a + (b * c)

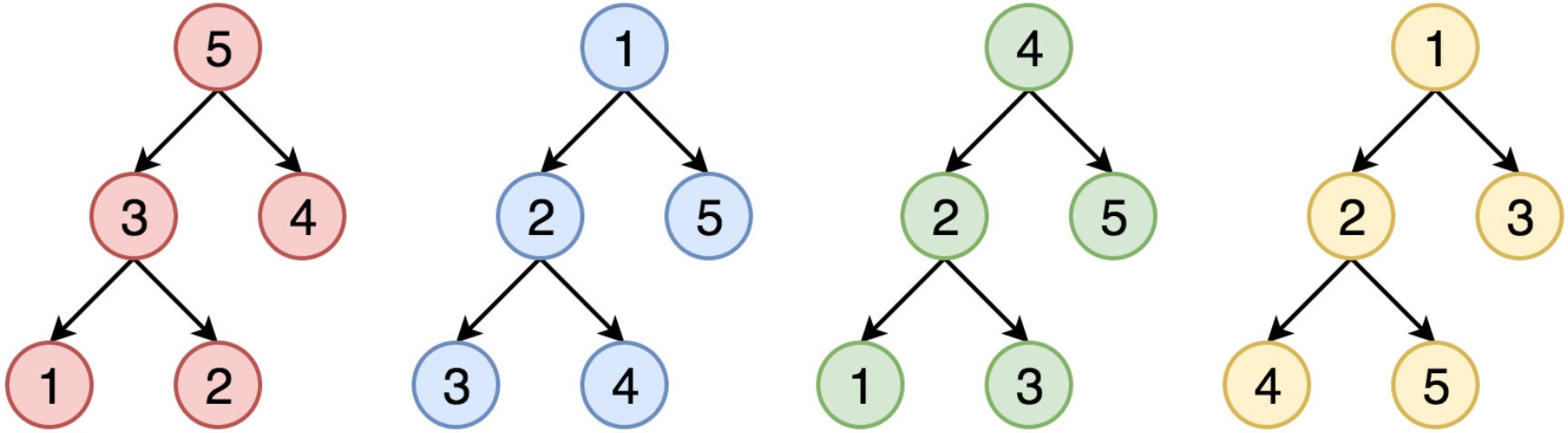
Outro exemplo

PREFIX	POSTFIX	INFIX
+ * * / a b c d e	a b / c * d * e +	a / b * c * d + e

- Como ler cada string e efetuar as operações ?
- Como usar o TAD STACK ?

Exemplo – Notação POSTFIX – Resultado ?

- 2 3 + 8 * = ?
- Criar STACK vazia
- Ler da esquerda para a direita
- Empilhar os operandos
- Sempre que se encontra um operador :
  - retirar os dois operandos que estão no topo da STACK
  - empilhar o resultado
- Façam este exemplo !!



[zhang-xiao-mu.blog]

Travessias de uma Árvore Binária

Travessias

- Visitar cada nó da árvore binária exatamente uma vez
- E efetuar algum tipo de processamento sobre cada nó
  - Imprimir
  - Alterar o valor
  - Escrever em ficheiro
  - …
- Vários tipos / ordens de travessia
- Travessias recursivas vs iterativas

Ordem da visita para o nó raiz e as subárvores

- Travessia em pré-ordem (NLR)

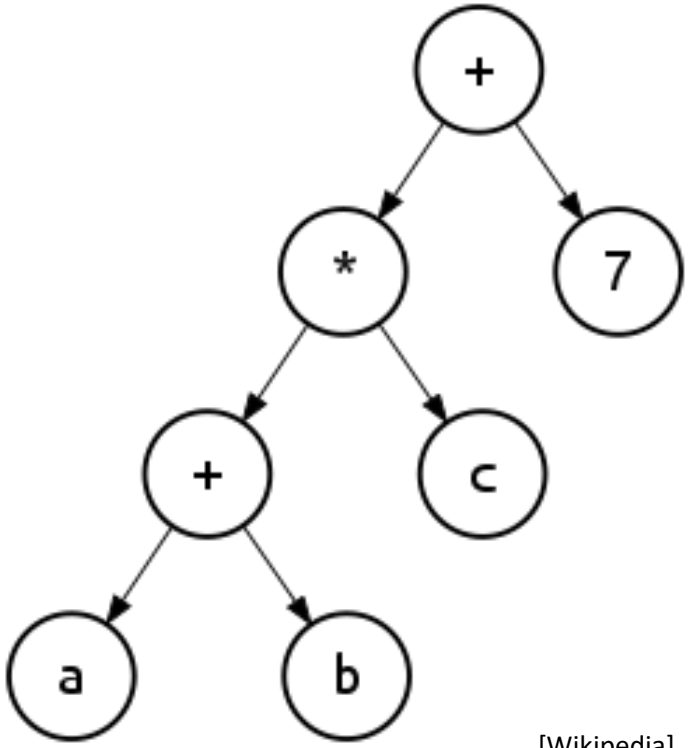
  + * + a b c 7

- Travessia em-ordem (LNR)

  a + b * c + 7

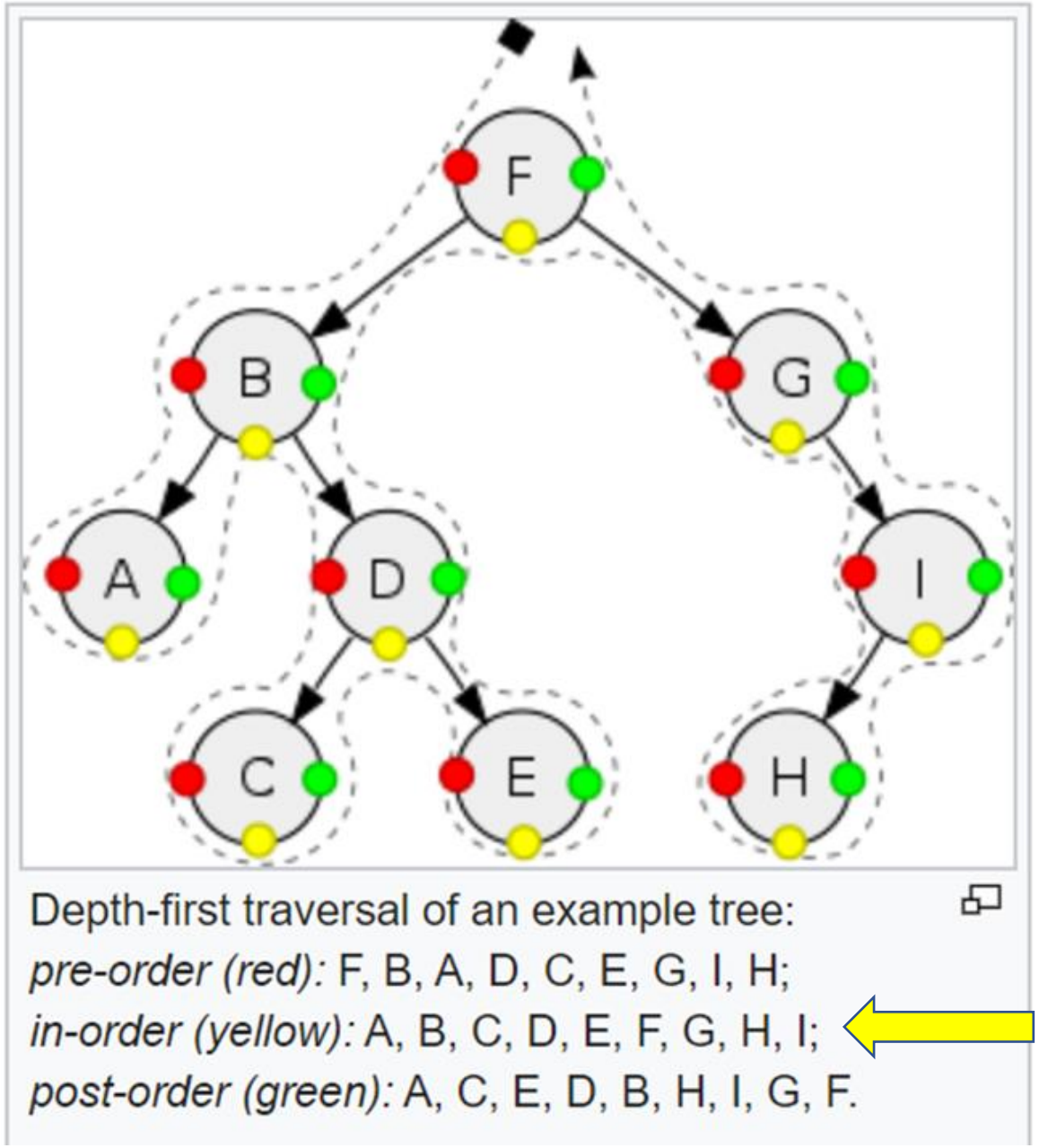
- Travessia em pós-ordem (LRN)

  a b + c * 7 +



[Wikipedia]

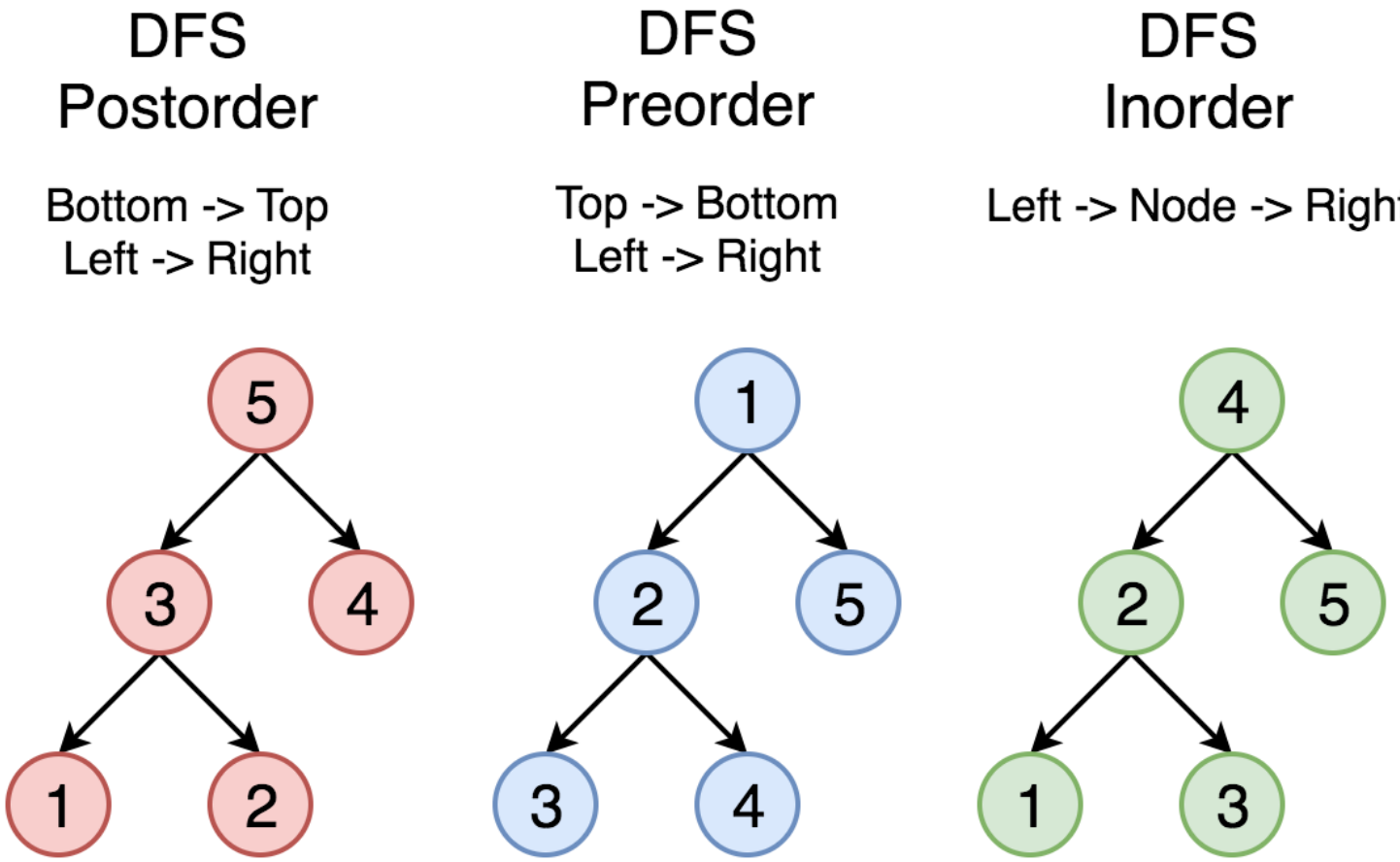
Travessias



Depth-first traversal of an example tree:
pre-order (red): F, B, A, D, C, E, G, I, H;
in-order (yellow): A, B, C, D, E, F, G, H, I;
post-order (green): A, C, E, D, B, H, I, G, F.

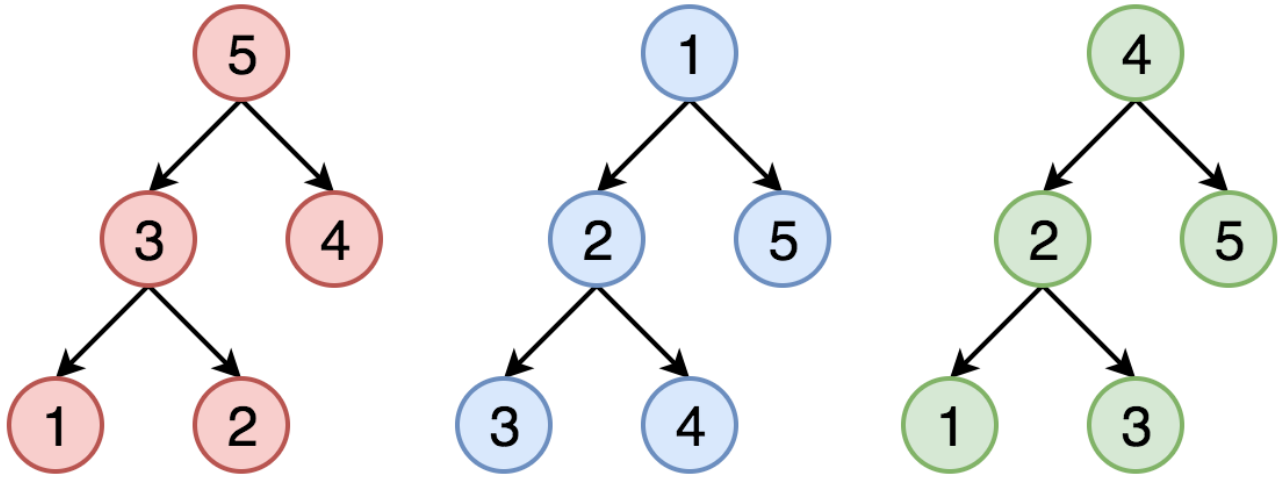
[Wikipedia]

Ordem / Travessias em profundidade



[zhang-xiao-mu.blog]

Travessias – Algoritmos Recursivos

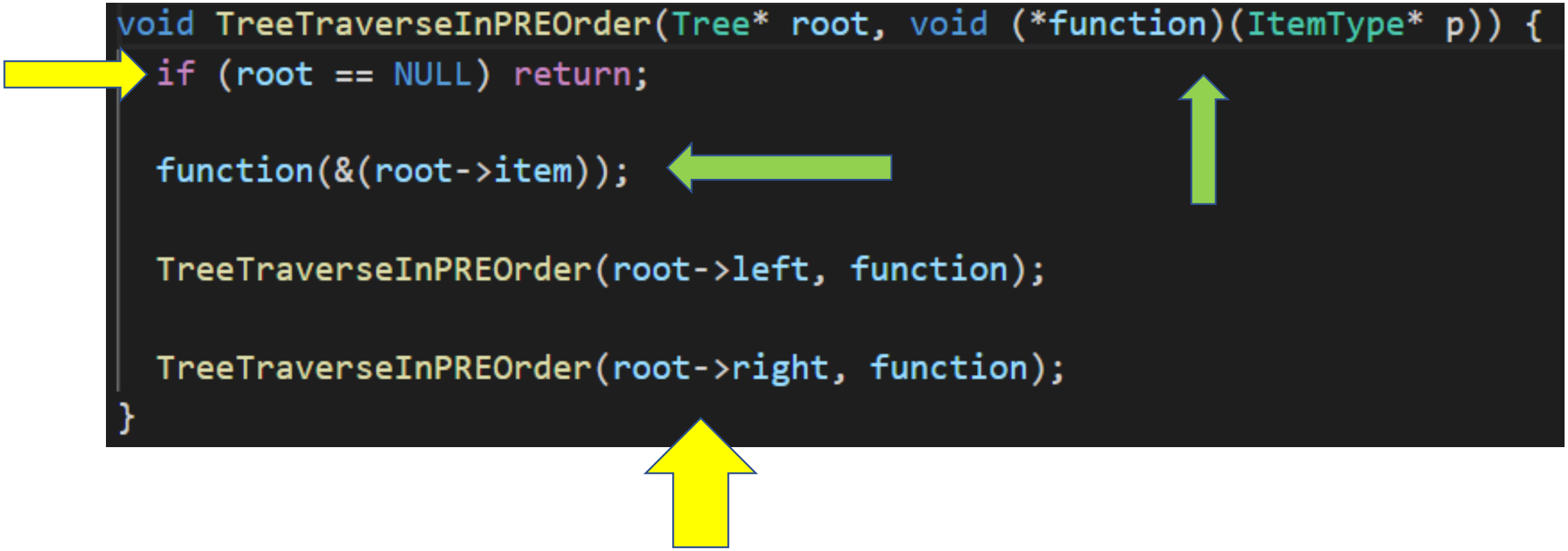


[zhang-xiao-mu.blog]

Travessias recursivas

- NLR – Pré-Ordem
  - processar o nó raiz
  - chamada recursiva para a subárvore esquerda
  - chamada recursiva para a subárvore direita
- LNR – Em-Ordem
  - chamada recursiva para a subárvore esquerda
  - processar o nó raiz
  - chamada recursiva para a subárvore direita
- LRN – Pós-Ordem
  - …

Travessia em pré-ordem

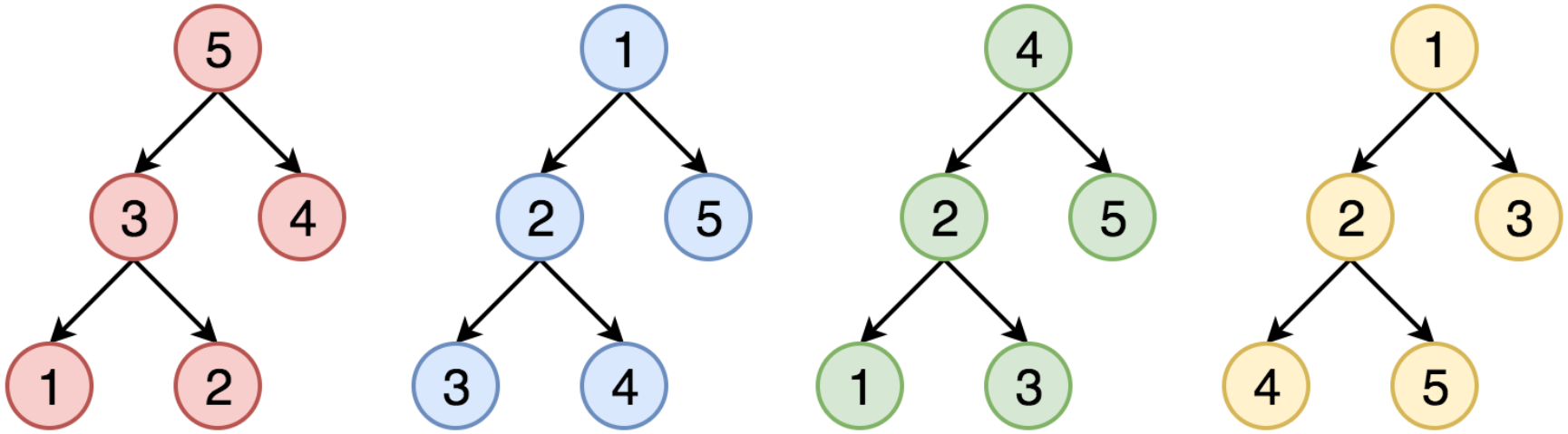


Exemplos de utilização – Funções genéricas

```c
void printInteger(int* p) { printf("%d ", *p); }

void multiplyIntegerBy2(int* p) { *p *= 2; }
```

```c
printf("PRE-Order traversal : ");

TreeTraverseInPREOrder(tree, printInteger);
```

```c
printf("Multiply each value by 2\n");

TreeTraverseInPREOrder(tree, multiplyIntegerBy2);
```

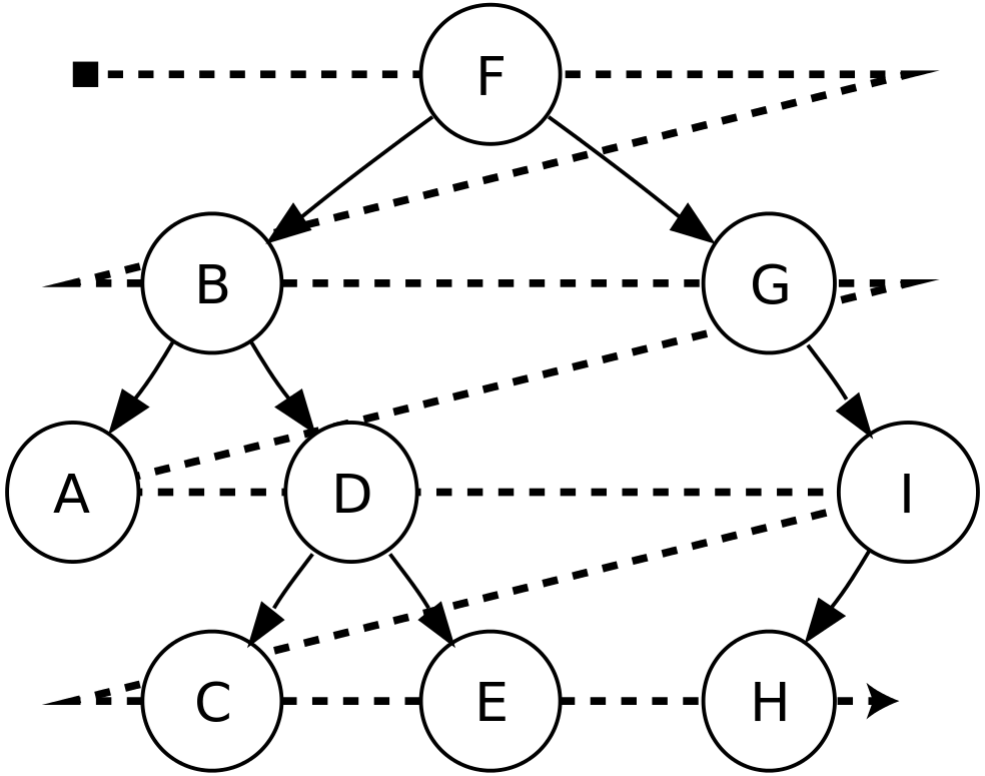


Travessias – Algoritmos Iterativos

[zhang-xiao-mu.blog]

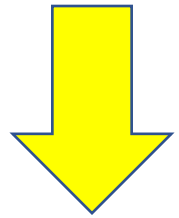
Uma travessia adicional – Travessia por níveis

- Mais um tipo de travessia
- Breadth-First traversal
- Como são visitados os nós da árvore ?
- A solução habitual usa uma FILA / QUEUE



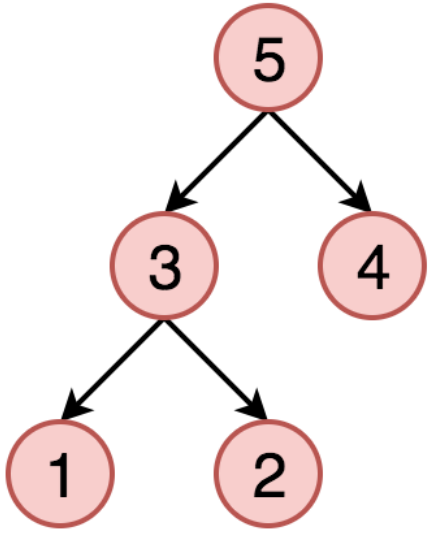
[Wikipedia]

Ordem / Travessias



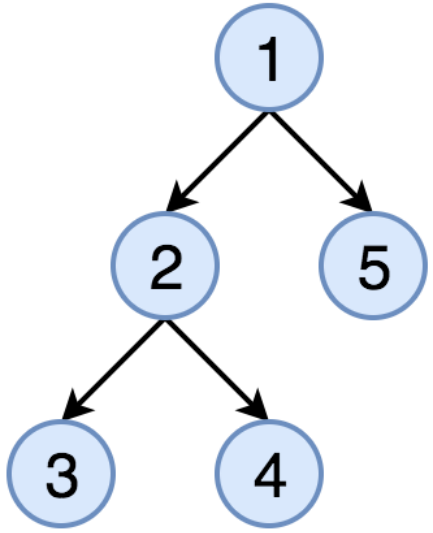
DFS
Postorder

Bottom -> Top
Left -> Right



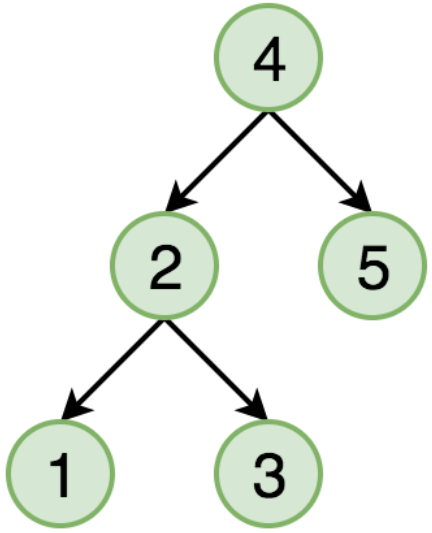
DFS
Preorder

Top -> Bottom
Left -> Right



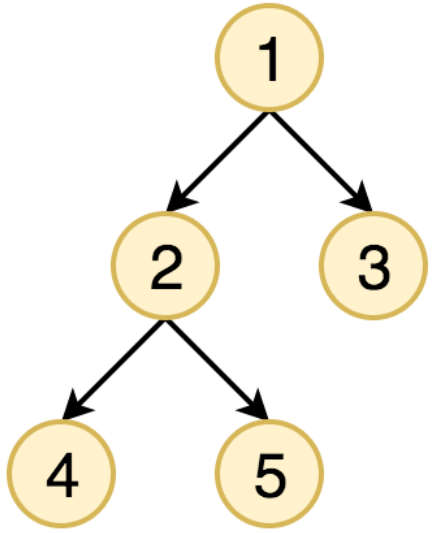
DFS
Inorder

Left -> Node -> Right



BFS

Left -> Right
Top -> Bottom



[zhang-xiao-mu.blog]

Travessias iterativas

- Usar uma estrutura de dados auxiliar : QUEUE ou STACK
- Armazenar ponteiros para os próximos nós a processar
  - Sequência de nós visitados ao longo da travessia
- QUEUE : Breadth-First – por níveis
- STACK : Depth-First – em profundidade
  - Pré-Ordem / Em-Ordem / Pós-Ordem

Estratégia básica

- Criar um conjunto vazio de ponteiros
- Adicionar o ponteiro para o nó raiz da árvore
- Enquanto o conjunto não for vazio

  Retirar do conjunto o ponteiro para o próximo nó 

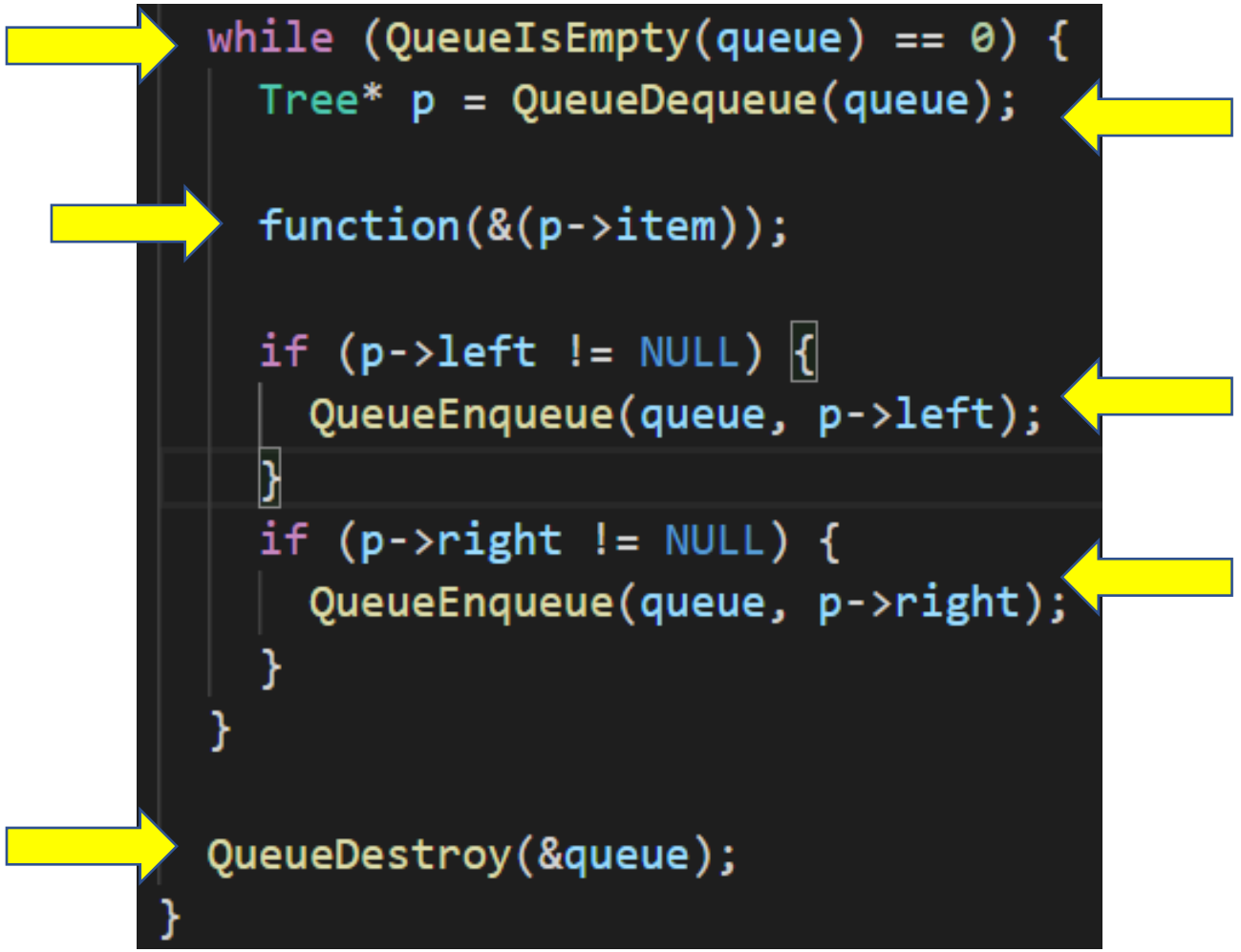
  Processar esse ponteiro / nó

  Se necessário, adicionar ponteiro(s) ao conjunto 
- Destruir o conjunto vazio

Travessia por níveis – QUEUE

```
void TreeTraverseLevelByLevelWithQUEUE(Tree* root,
                                       void (*function)(ItemType* p)) {
  if (root == NULL) {
    return;
  }

  // Not checking for queue errors !!
  // Create the QUEUE for storing POINTERS

  Queue* queue = QueueCreate();

  QueueEnqueue(queue, root);
```

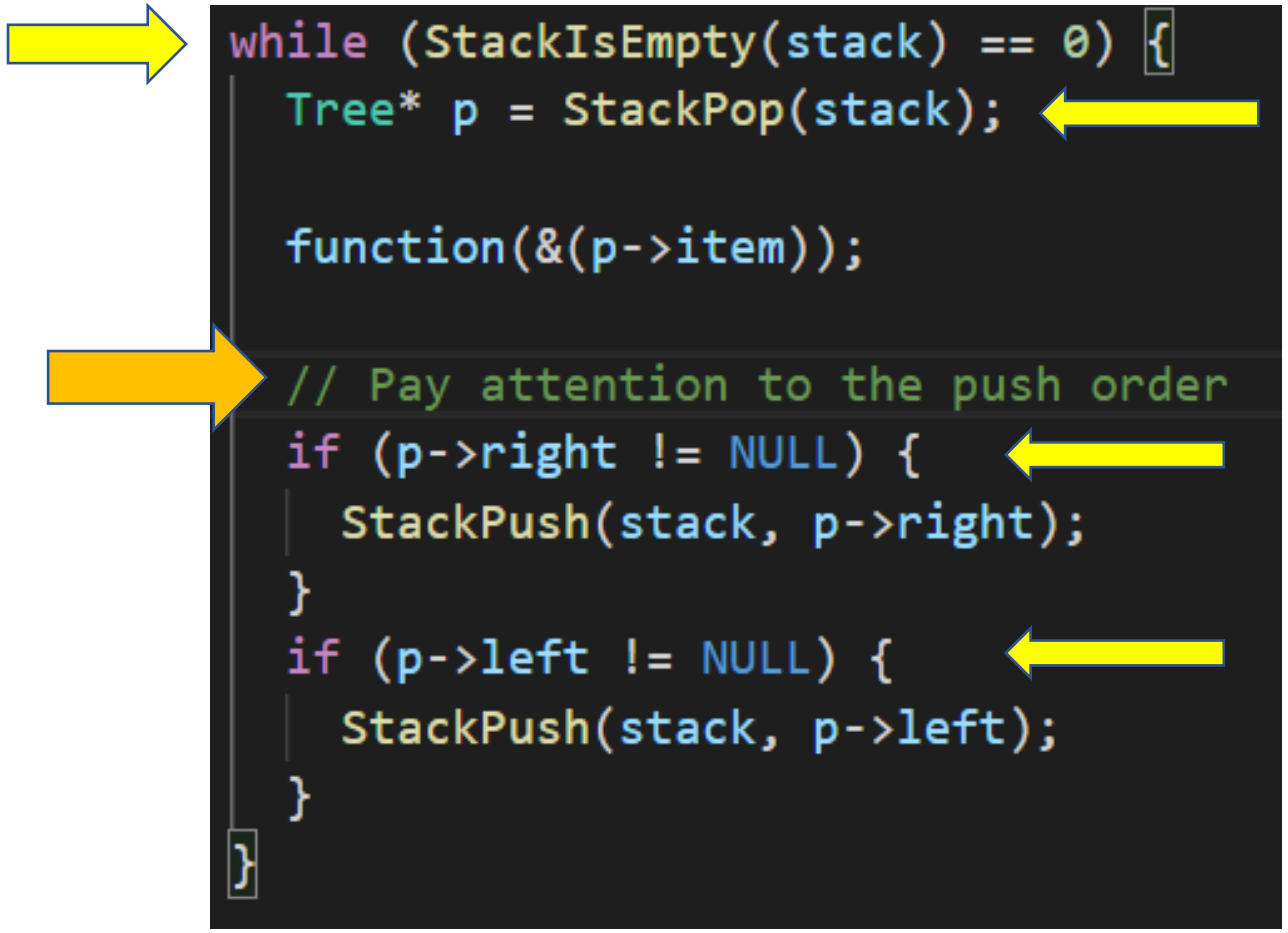
Travessia por níveis – QUEUE



```
while (QueueIsEmpty(queue) == 0) {
  Tree* p = QueueDequeue(queue);

  function(&(p->item));

  if (p->left != NULL) {
    QueueEnqueue(queue, p->left);
  }
  if (p->right != NULL) {
    QueueEnqueue(queue, p->right);
  }
}

QueueDestroy(&queue);
}
```

- Enquanto houver nós não visitados
- Processar o próximo nó
- Se houver um filho esquerdo, adicionar o seu ponteiro à QUEUE
- Se houver um filho direito, adicionar o seu ponteiro à QUEUE

Travessia em Pré-Ordem – STACK

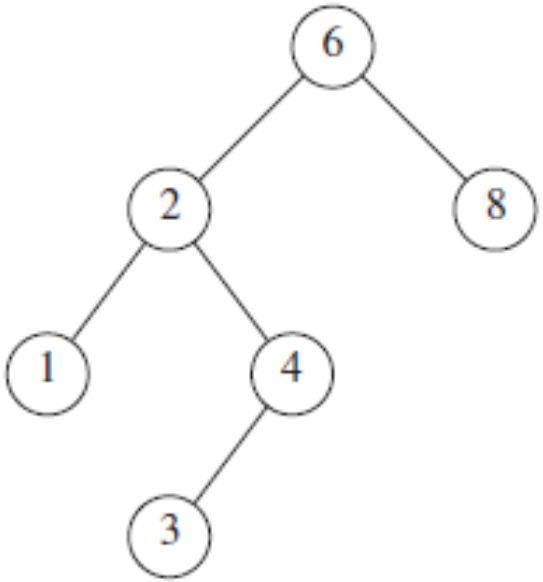


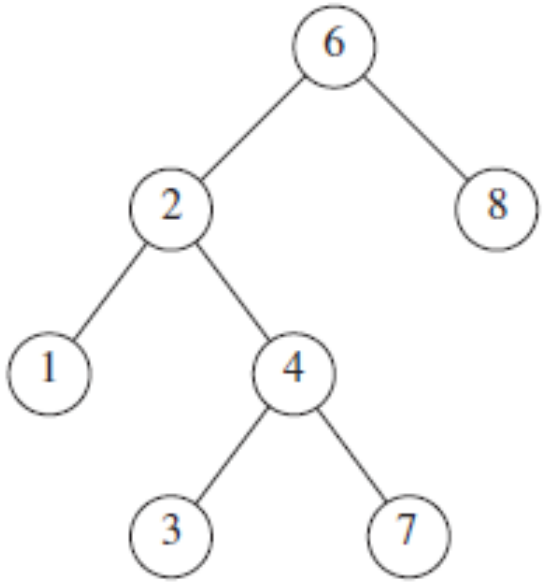
```
while (StackIsEmpty(stack) == 0) {
  Tree* p = StackPop(stack);

  function(&(p->item));

  // Pay attention to the push order
  if (p->right != NULL) {
    StackPush(stack, p->right);
  }
  if (p->left != NULL) {
    StackPush(stack, p->left);
  }
}
```

- Enquanto houver nós não visitados
- Processar o próximo nó
- Se houver um filho direito, adicionar o seu ponteiro à STACK
- Se houver um filho esquerdo, adicionar o seu ponteiro à STACK

Árvores Binárias de Procura (ABP) – Binary Search Trees (BST)

Critério de ordem ?





- Qual das árvores está ordenada ?
- Que operações são mais eficientes por existir uma ordem ?

TAD Árvore Binária de Procura

- Conjunto de elementos do mesmo tipo
- Armazenados **em-ordem**
- Que operações beneficiam / dependem da ordem dos elementos ?

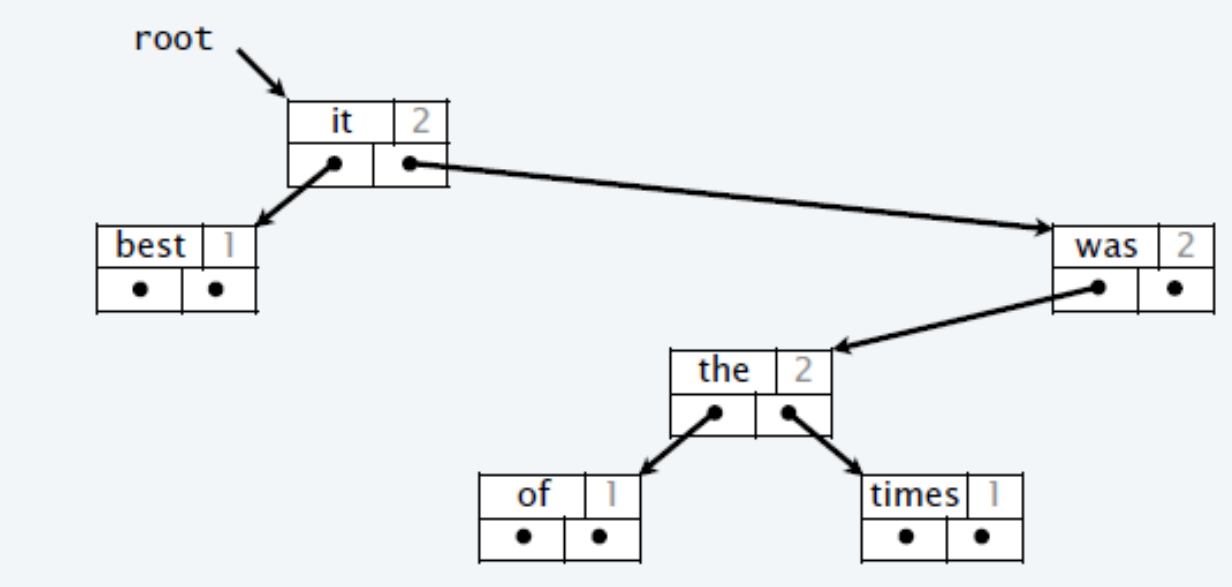
- Procura / inserção / remoção / substituição
- Pertença

- **search()** / **insert()** / **remove()** / **replace()**
- **contains()**
- size() / isEmpty()
- create() / destroy()

Critério de ordem – Definição recursiva

- Para cada nó, os elementos da sua **subárvore esquerda** são **inferiores** a esse nó
- E os elementos da sua **subárvore direita** são **superiores** a esse nó
- **Não** há elementos **repetidos** !!

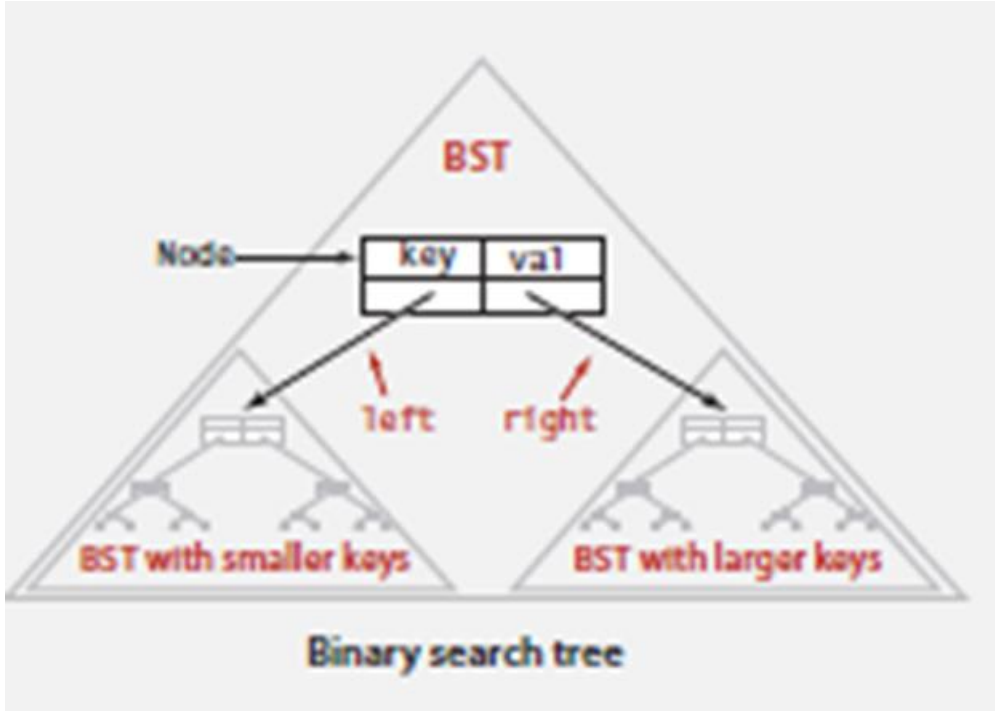
- A organização da árvore depende da **sequência de inserção** dos elementos



[Sedgewick & Wayne]

- Qual é a ordem de inserção dos nós nesta árvore ?

Critério de ordem – Definição recursiva

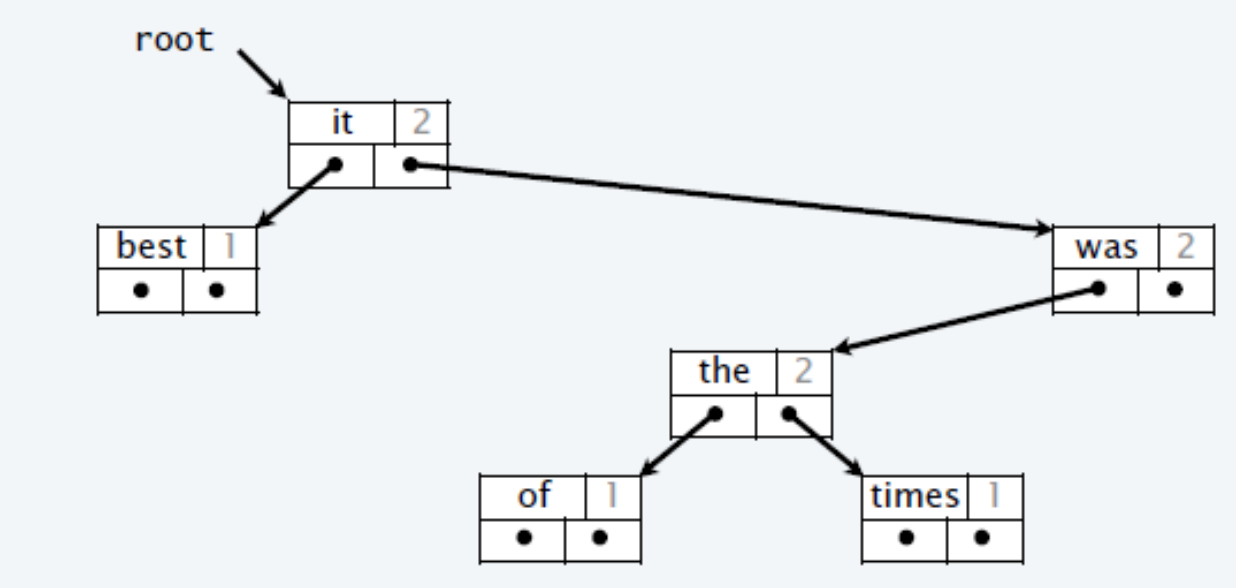


[Sedgewick & Wayne]

Operações habituais

- O item armazenado em cada nó é, em geral, um par (chave, valor)
- Ordem determinada pelo valor das chaves

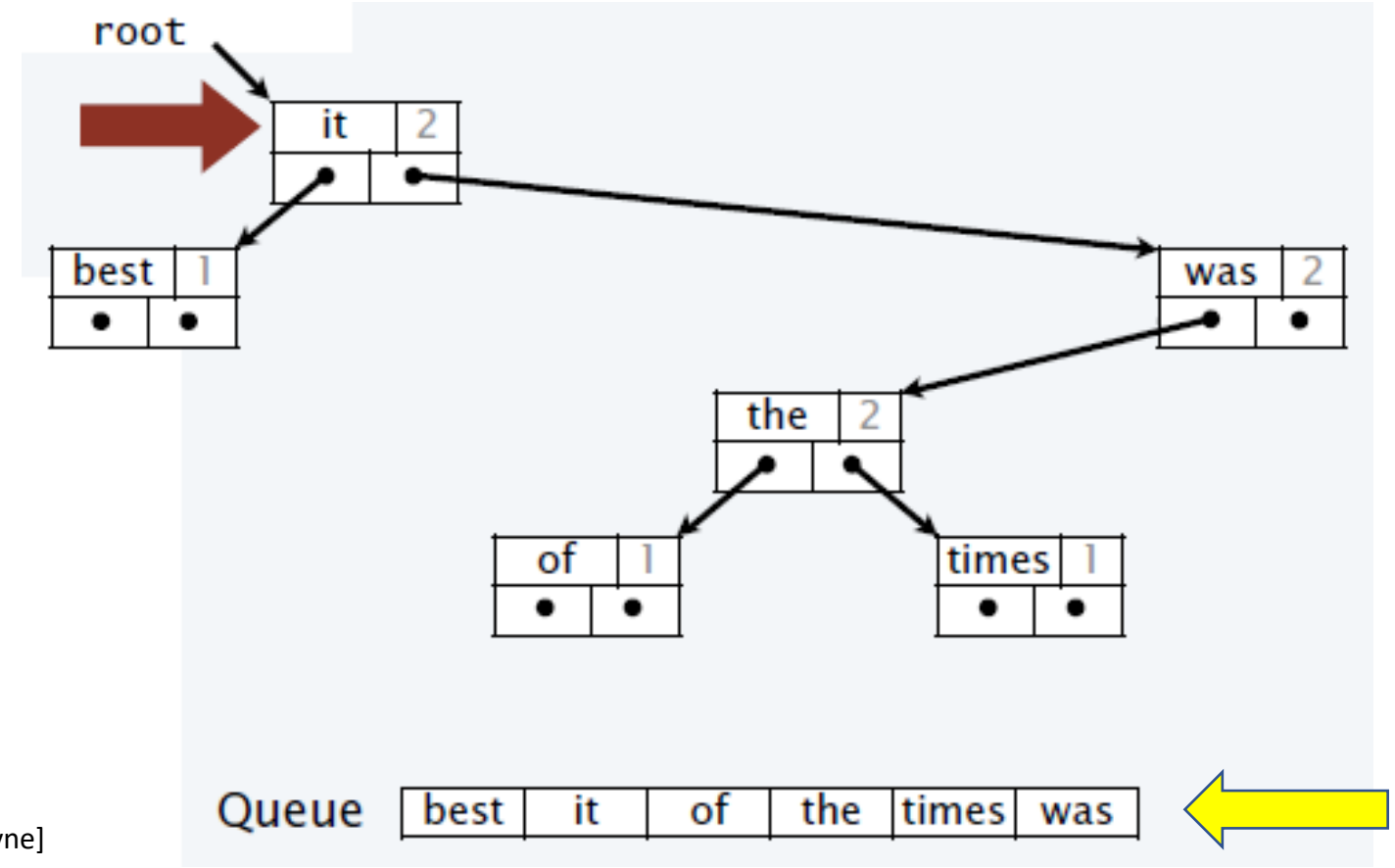
- Procurar
- Adicionar
- Alterar
- Remover
- Visitar em-ordem



[Sedgewick & Wayne]

Travessia em-ordem

- Exemplo : preencher uma fila com os itens ordenados

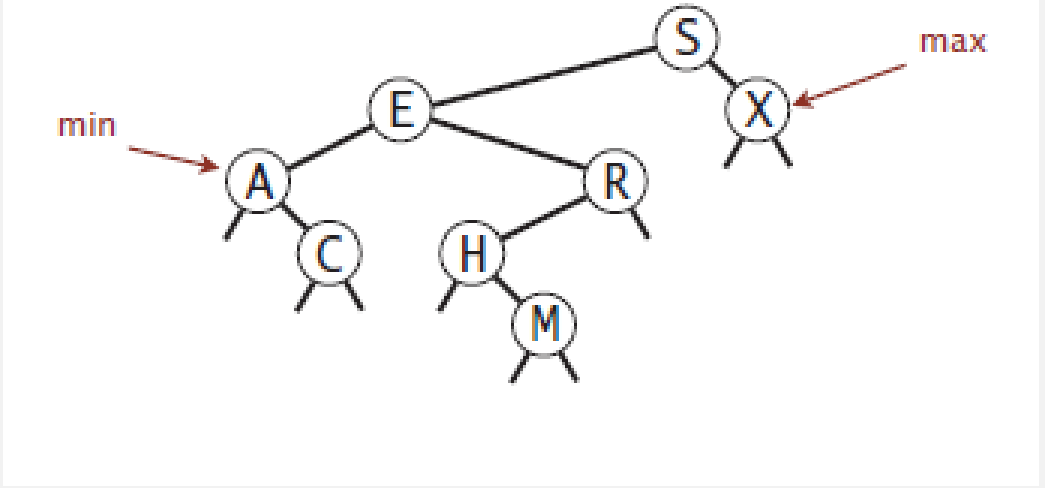


[Sedgewick & Wayne]

Árvores Binárias de Procura (ABP) – Algumas funções

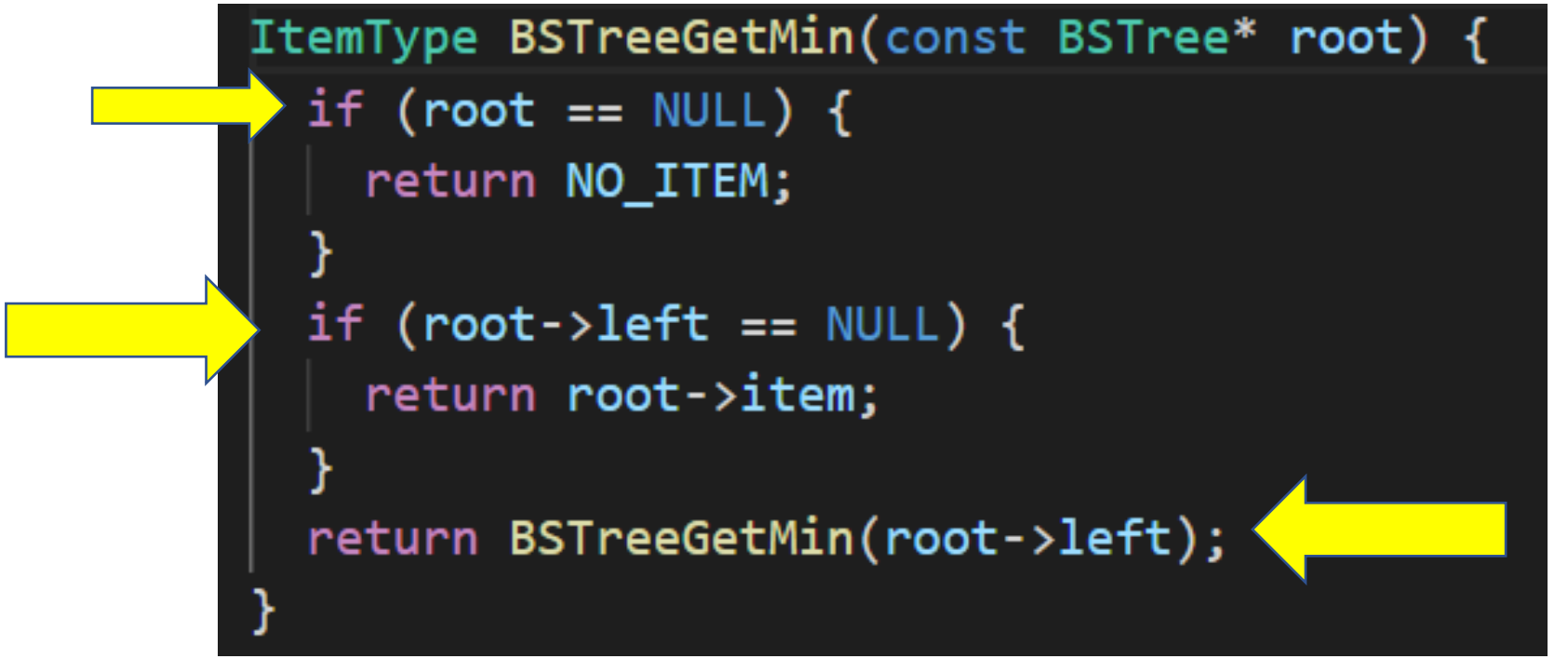
ABP – Menor elemento ? / Maior elemento ?

- Onde estão ?
- Como fazer ?
- Eficiência ?



[Sedgewick & Wayne]

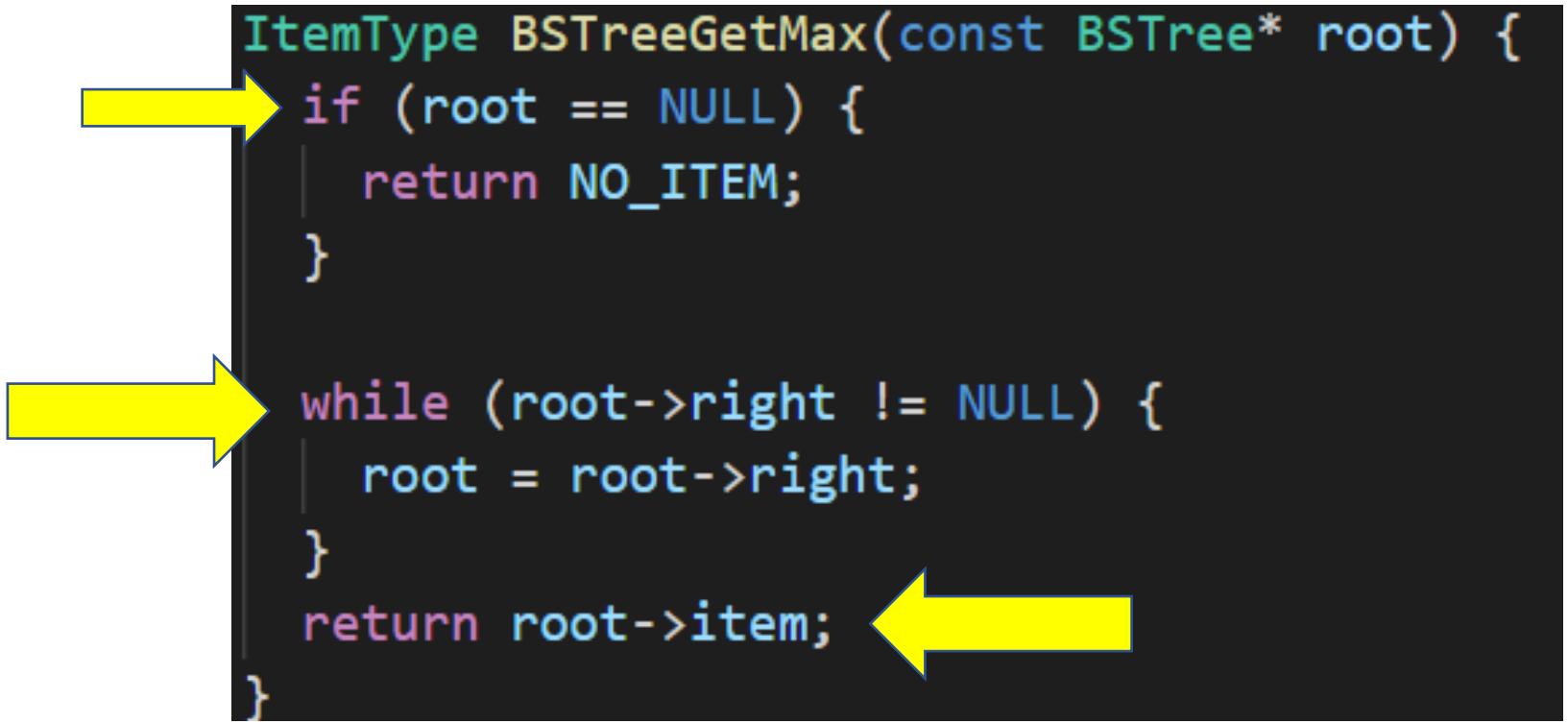
ABP – getMin() – Versão recursiva



```
ItemType BSTreeGetMin(const BSTree* root) {
  if (root == NULL) {
    return NO_ITEM;
  }
  if (root->left == NULL) {
    return root->item;
  }
  return BSTreeGetMin(root->left);
}
```

- Procurar o nó “mais à esquerda”, i.e., que não tem filho esquerdo

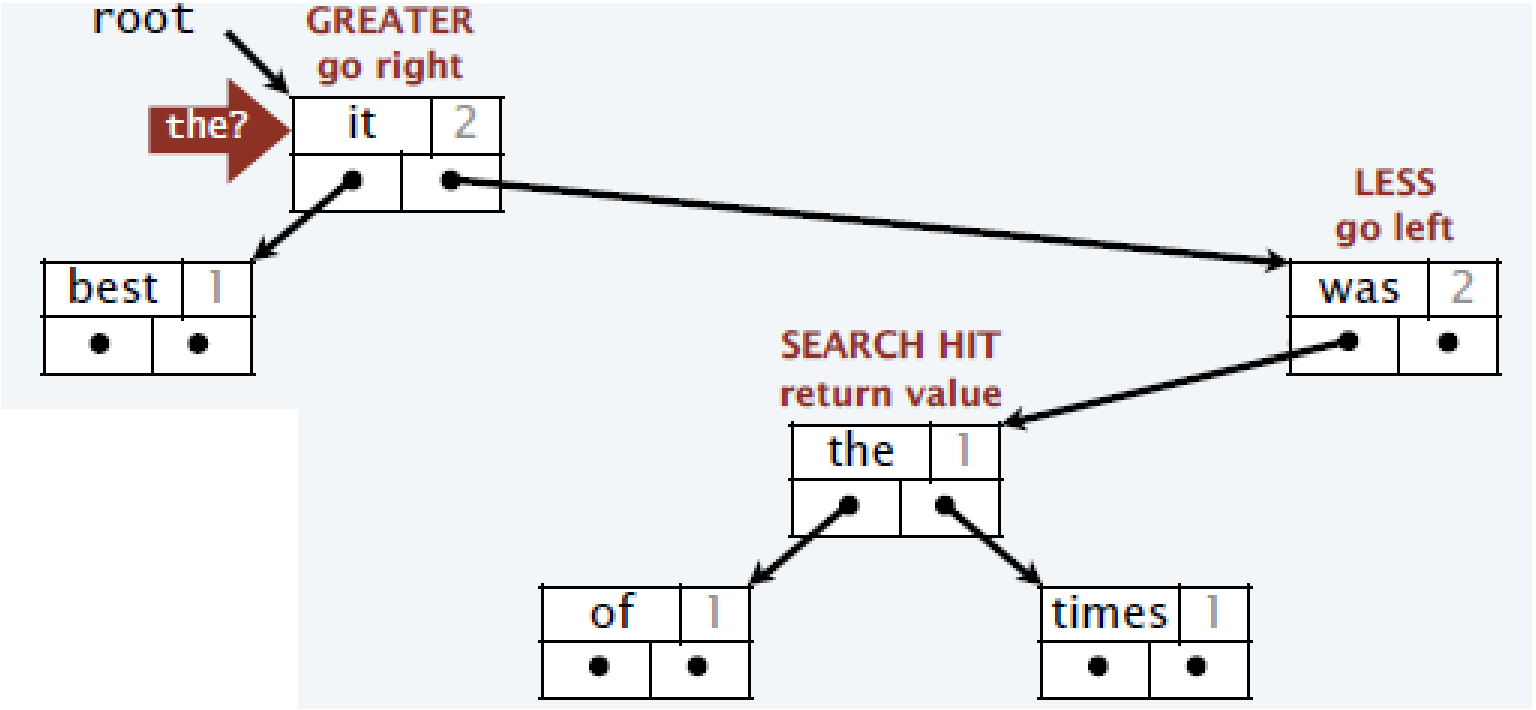
ABP – getMax() – Versão iterativa



```
ItemType BSTreeGetMax(const BSTree* root) {
  if (root == NULL) {
    return NO_ITEM;
  }

  while (root->right != NULL) {
    root = root->right;
  }
  return root->item;
}
```

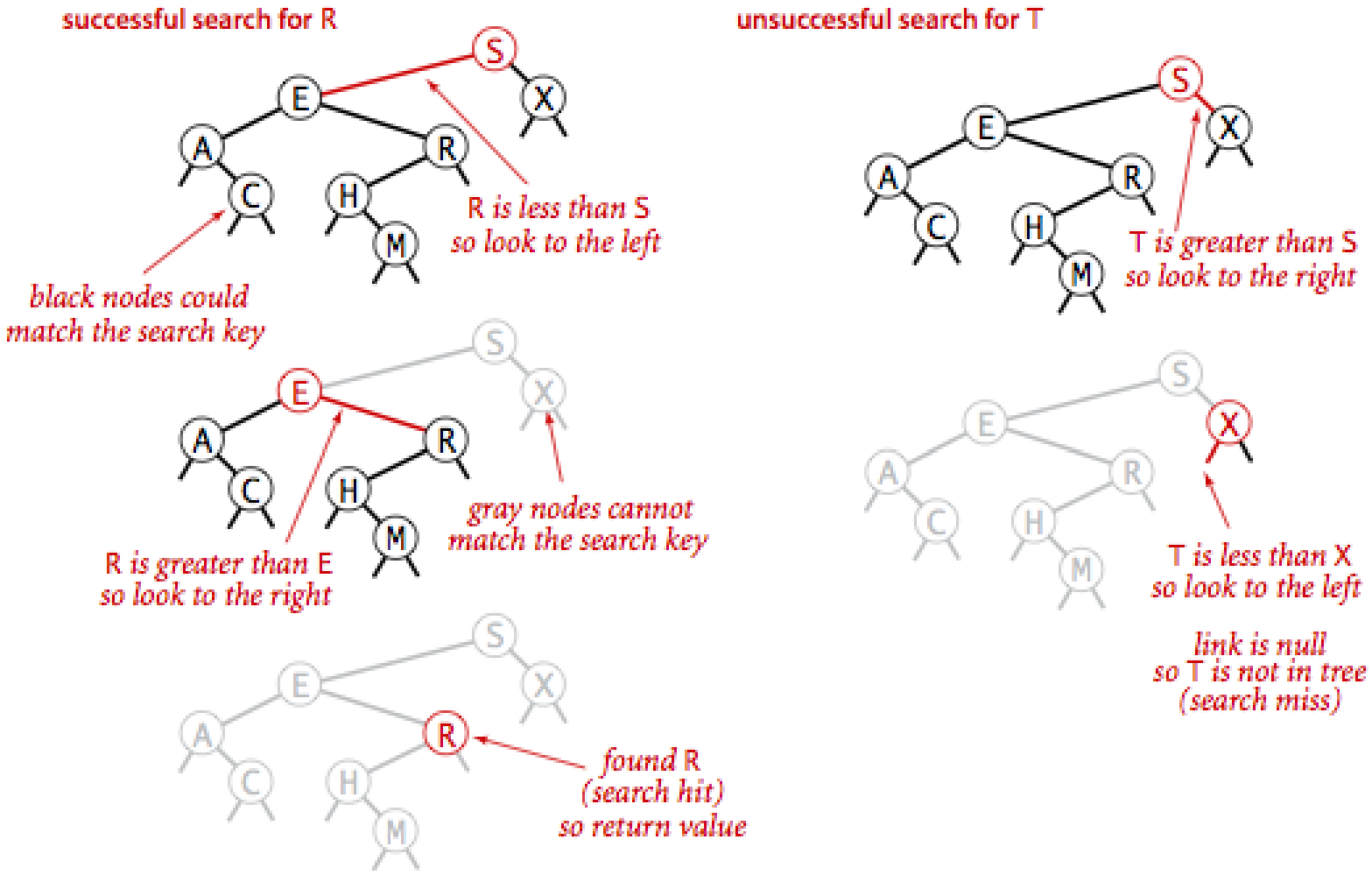
- Procurar o nó “mais à direita”, i.e., que não tem filho direito

ABP – Procurar um elemento – Como fazer ?



[Sedgewick & Wayne]

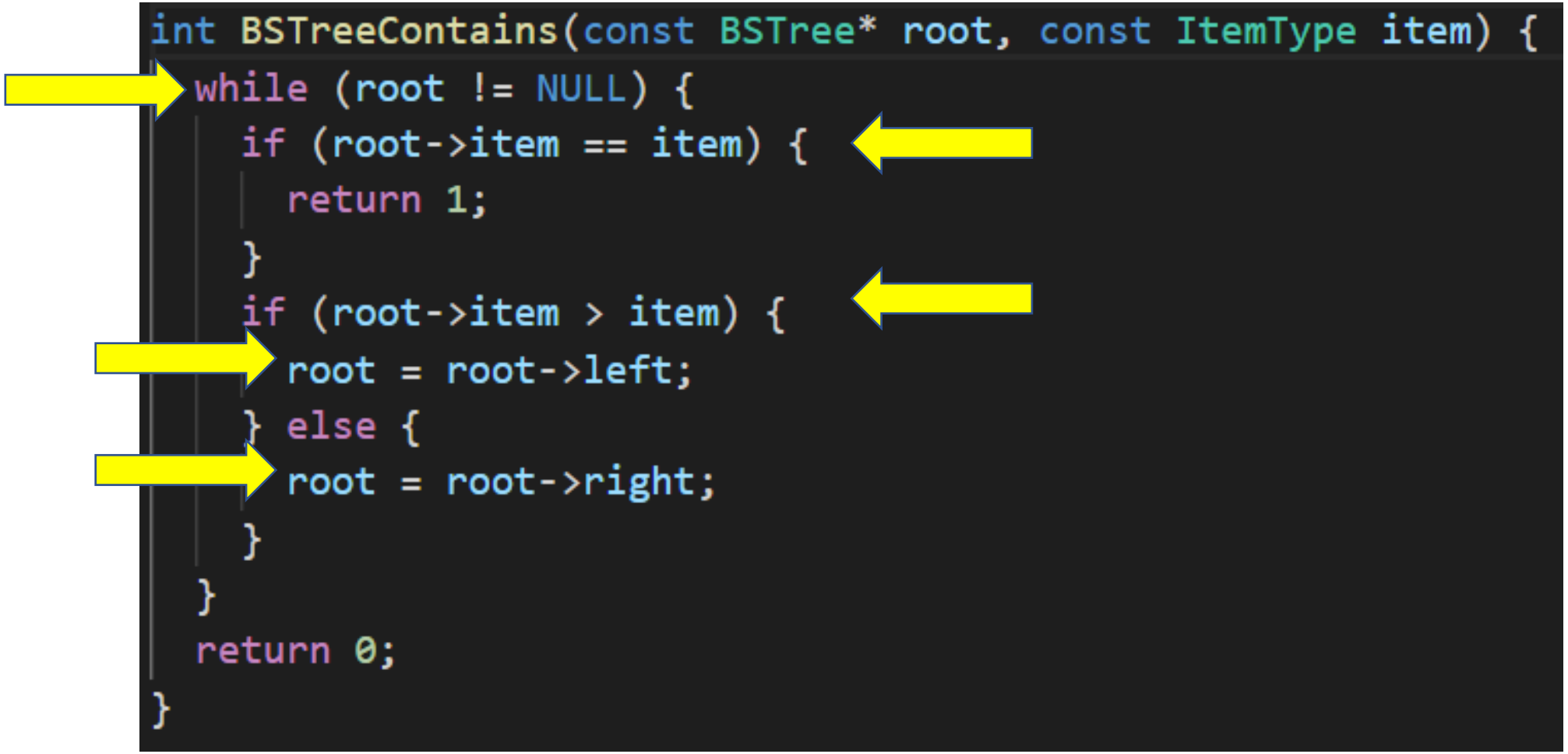
Exemplos



Successful (left) and unsuccessful (right) search in a BST

[Sedgewick & Wayne]

ABP – Procurar – Versão iterativa



```
int BSTreeContains(const BSTree* root, const ItemType item) {
  while (root != NULL) {
    if (root->item == item) {
      return 1;
    }
    if (root->item > item) {
      root = root->left;
    } else {
      root = root->right;
    }
  }
  return 0;
}
```

Exercícios / Tarefas

Ex. 1 : Escrever a expressão nas 3 notações

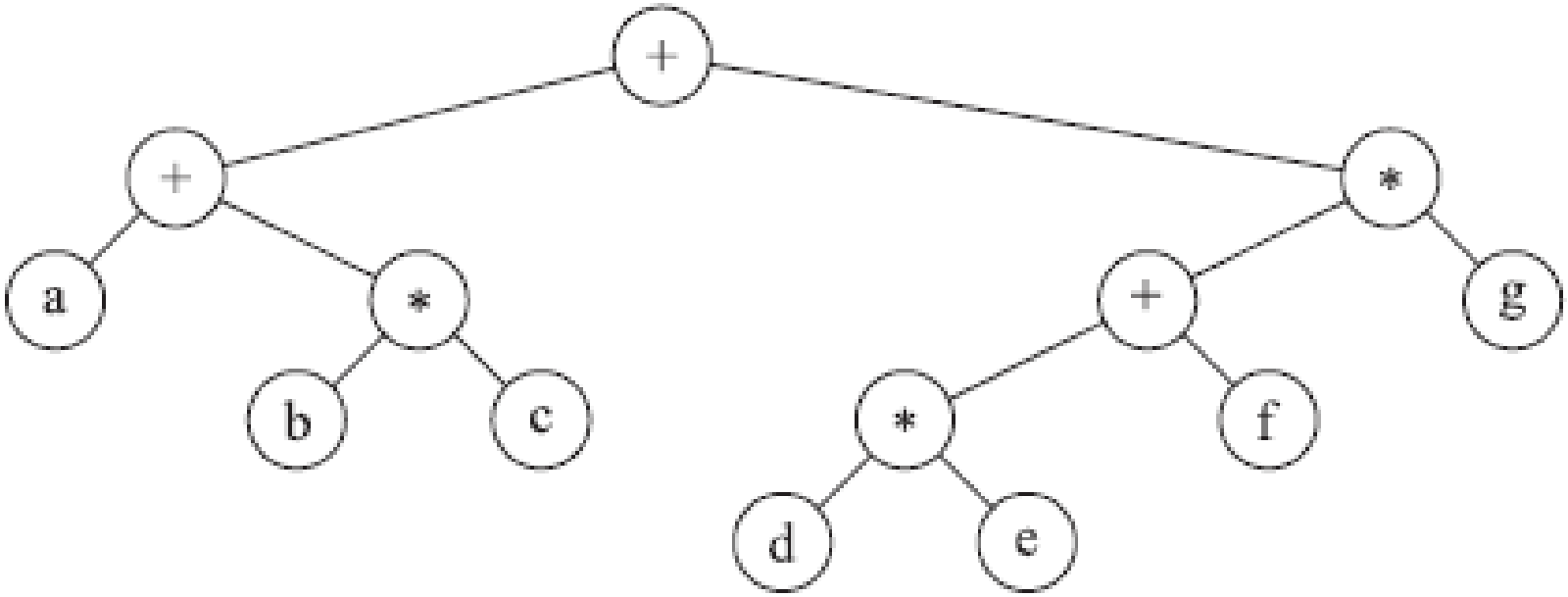


Figure 4.14 Expression tree for (a + b * c) + ((d * e + f) * g)

[Weiss]

Exercício 2 – Verdadeiro ou Falso

- Na travessia em **Pré-Ordem** de uma árvore binária, todos os elementos da subárvore direita da raiz são visitados **primeiro** que os elementos da subárvore esquerda da raiz.

- Na travessia em **Pós-Ordem** de uma árvore binária, todos os elementos da subárvore esquerda da raiz são visitados **primeiro** que os elementos da subárvore direita da raiz.

- Numa **Árvore Binária de Procura (*"Binary Search Tree"*)**, a sub-árvore direita de um dado nó pode conter elementos de valor inferior a esse mesmo nó.

Tarefa 1 : ABP – getMin() – Versão iterativa

- Desenvolva uma função iterativa que devolva o valor do menor elemento pertencente a uma Árvore Binária de Procura (ABP) / Binary Search Tree (BST) que armazena números inteiros

Tarefa 2 : ABP – getMax() – Versão recursiva

- Desenvolva uma função recursiva que devolva o valor do maior elemento pertencente a uma Árvore Binária de Procura (ABP) / Binary Search Tree (BST) que armazena números inteiros
- Não use variáveis globais !!

Tarefa 3 : ABP – Procurar – Versão recursiva

- Desenvolva uma função recursiva que, dado um valor inteiro e uma Árvore Binária de Procura (ABP) / Binary Search Tree (BST), devolva um ponteiro para o nó armazenando esse valor, caso exista na árvore, ou o ponteiro NULL, caso contrário

- Não use variáveis globais !!

Tarefa 4 : Implementar as travessias recursivas

- Travessia em pré-ordem
- Travessia em-ordem
- Travessia em pós-ordem

- Atenção à ordem de visita das subárvores !!
- Listar os elementos de uma árvore e confirmar a ordem

Tarefa 5 : Travessia em pré-ordem – Aplicação

- Analisar o código
- Registar a informação de uma árvore num ficheiro, usando uma travessia em pré-ordem
- Recuperar a informação de uma árvore a partir de um ficheiro, usando uma travessia em pré-ordem

Tarefa 6 : Travessias iterativas

- Analisar o código
- Travessia iterativa EM-ORDEM, usando uma PILHA / STACK
- Travessia iterativa em PÓS-ORDEM, usando uma PILHA / STACK