

# Árvores Binárias I

Joaquim Madeira

28/04/2020

# Ficheiro ZIP

- Está disponível no Moodle um **ficheiro ZIP** de suporte aos tópicos de hoje
- **1ª versão** do tipo abstrato **Árvore Binária**
- **Funções incompletas**, que permitem trabalho autónomo de desenvolvimento e teste

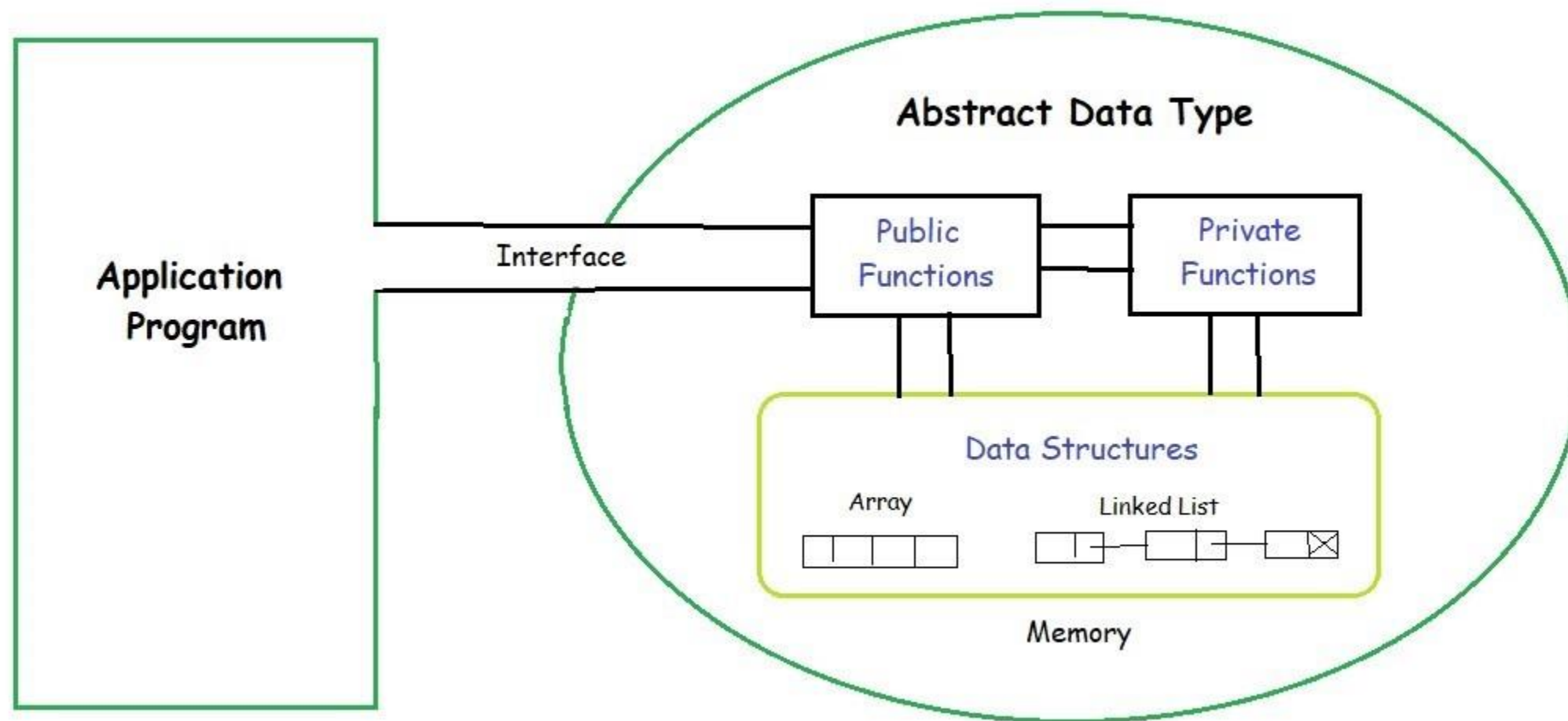
# Sumário

- Recap
- Árvores binárias; terminologia e algumas propriedades
- O TAD Árvore Binária
- Possíveis estruturas de dados
- Algoritmos recursivos – exemplos simples

Let's  
RECAP

# Recapitulação

# Tipo Abstrato de Dados (TAD)

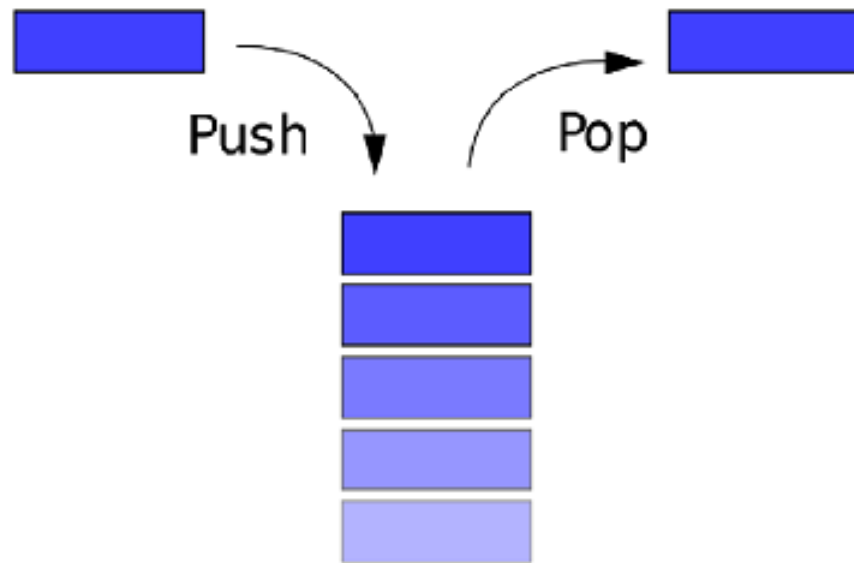


[geeksforgeeks.org]

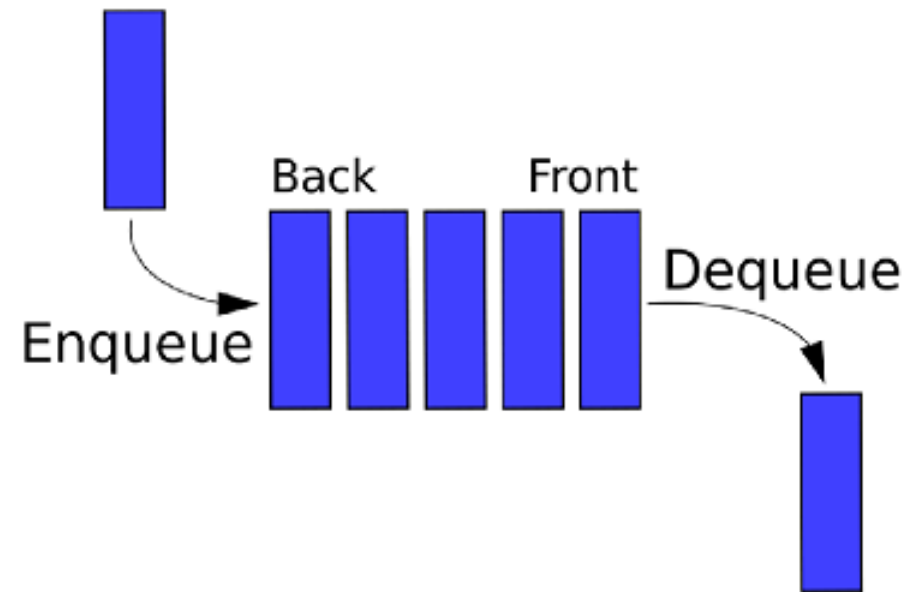
# Tipo Abstrato de Dados (TAD)

- TAD = especificação + interface + implementação
- Encapsular detalhes da representação / implementação
- Flexibilizar manutenção / reutilização / portabilidade
  
- Ficheiro .h : operações públicas + ponteiro para instância
- Ficheiro .c : implementação + representação interna

# STACK e QUEUE



Stack (LIFO) last in first out



Queue (FIFO) first in first out

[github.io]

- Num próximo **guião prático** iremos usar / aplicar

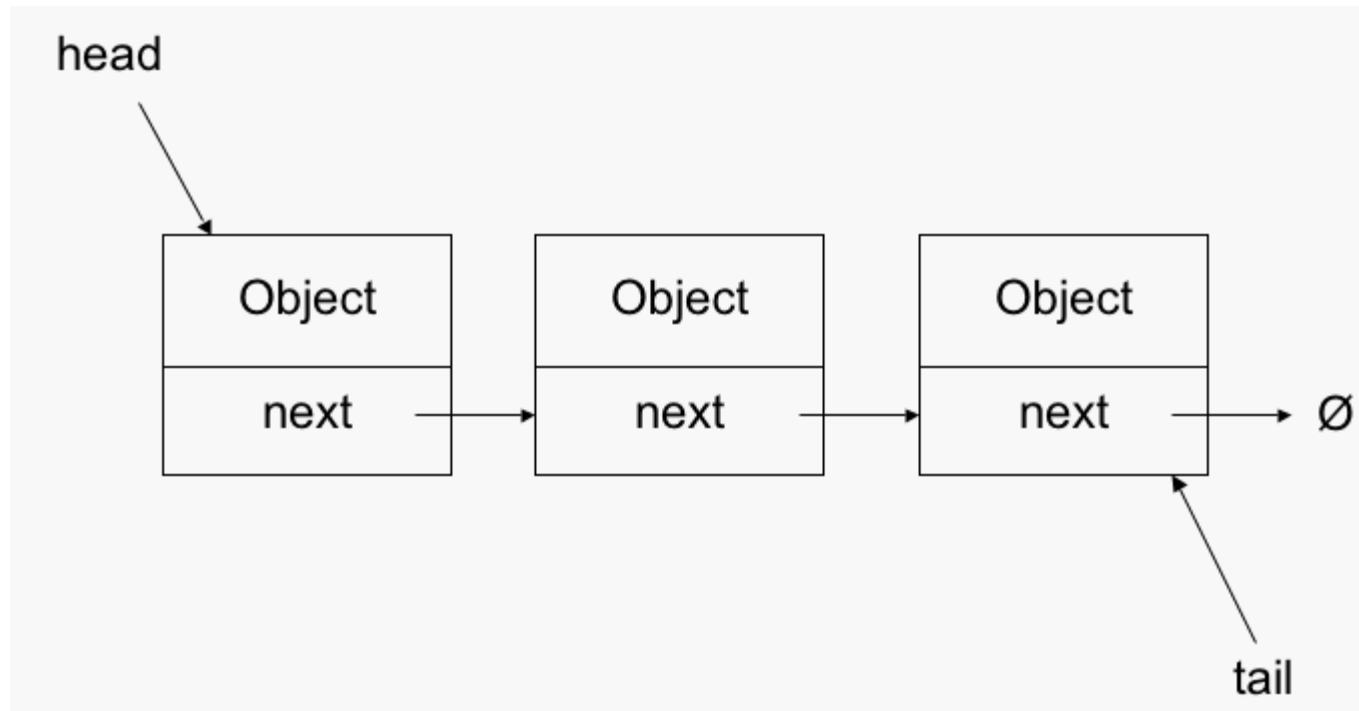
# DEQUE – Tentaram fazer ? – Questões ?



[java2novice.com]



# LINKED LIST – Tentaram fazer ? – Questões ?



[usfCS]

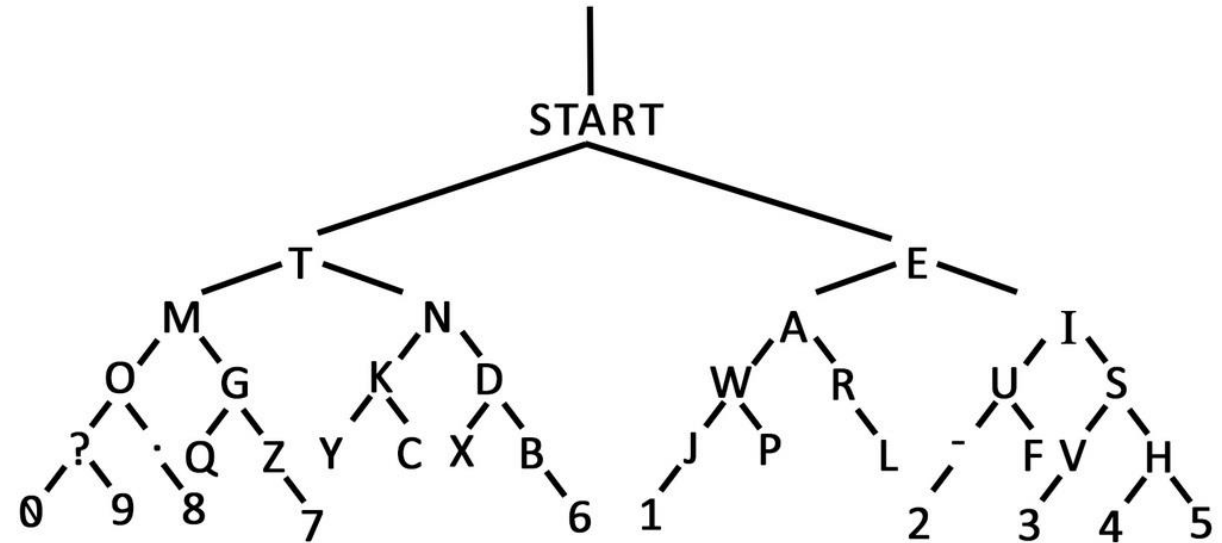
- Num próximo **guião prático** iremos analisar / completar / aplicar

← DASH

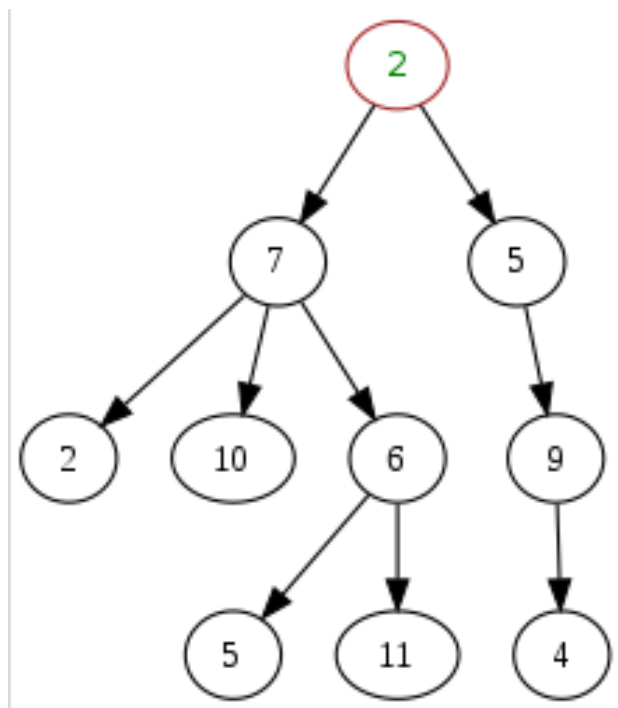
DOT →

[weebly.com]

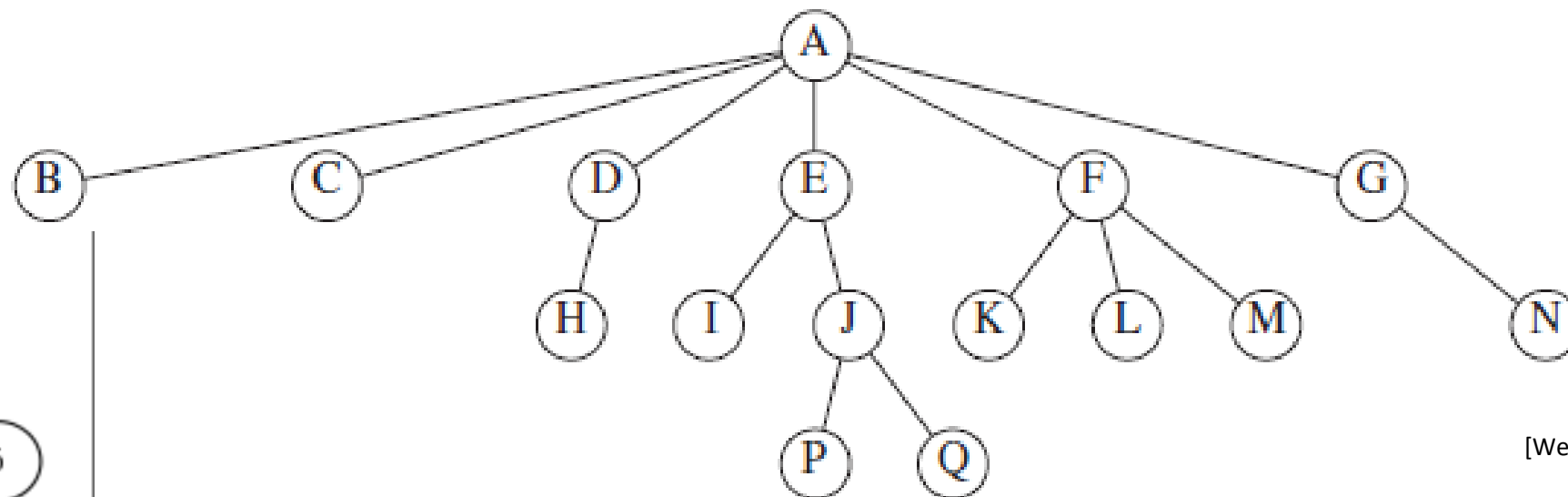
# Árvores



# Árvores – Arcos ? – Ordem ?



[Wikipedia]

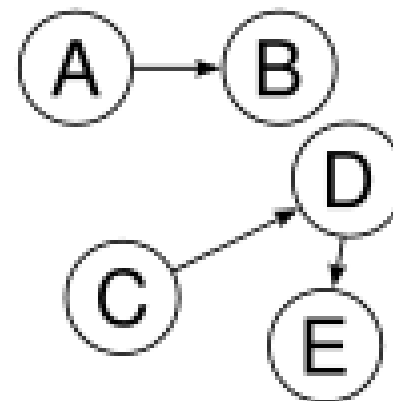
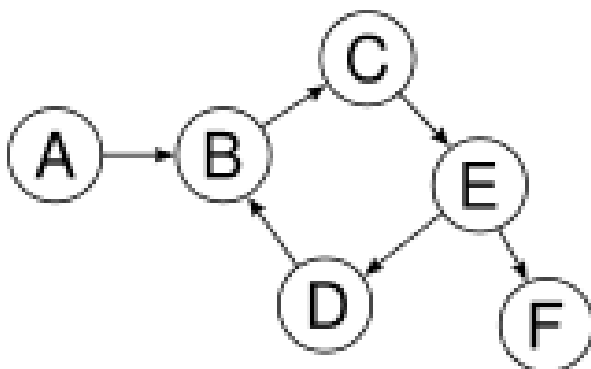
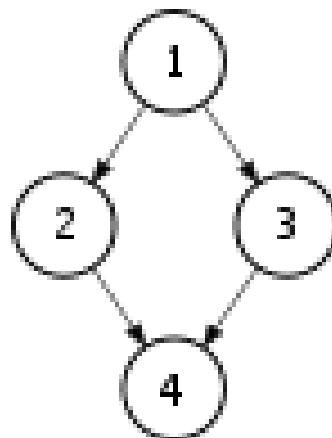
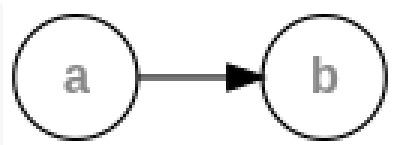


[Weiss]

# Árvores vs Grafos

- Todas as **árvores** são **grafos** !
- **MAS**, nem todos os grafos são árvores !!
- Árvore: existe **um só caminho** entre qualquer **par de nós distintos**
- Árvore: **apagar** um qualquer arco origina duas árvores **desconexas**
- Árvore com  **$n$  nós** tem  **$(n - 1)$  arcos** e não contém nenhum ciclo
- Árvore com  **$n$  nós** tem  **$(n - 1)$  arcos** e é conexa

# São árvores orientadas ou **não** ?



[Wikipedia]

# Árvores orientadas vs Grafos orientados

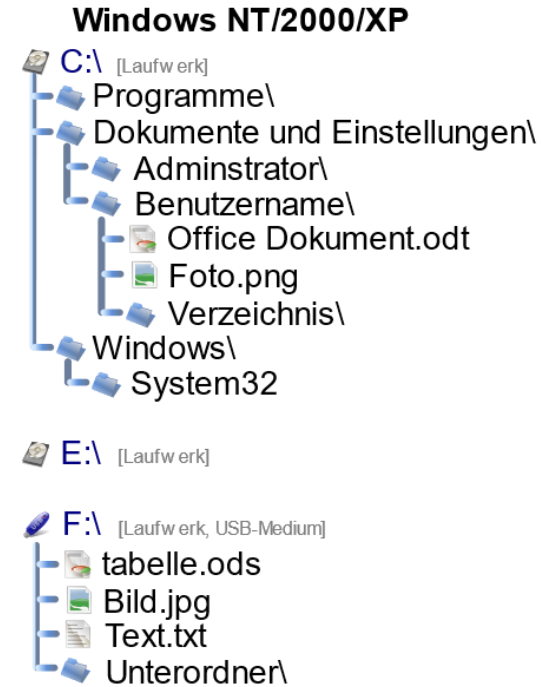
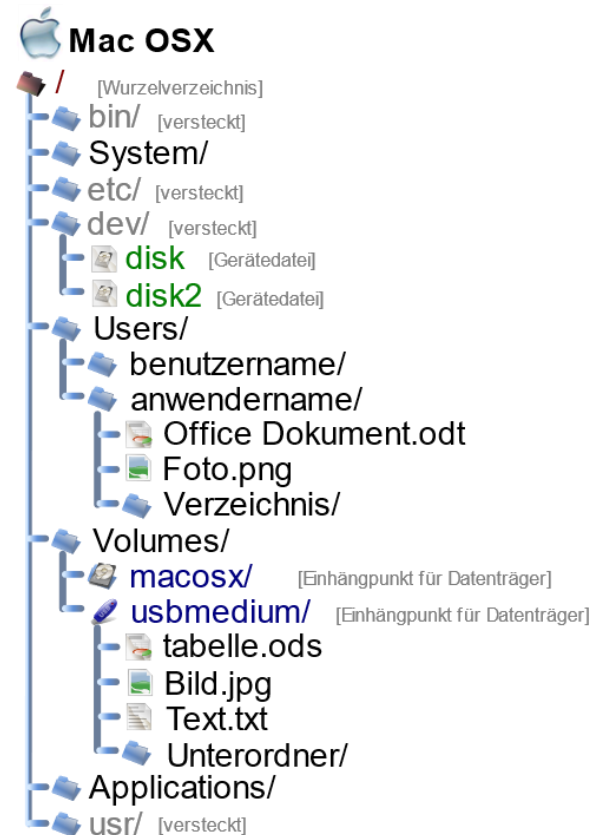
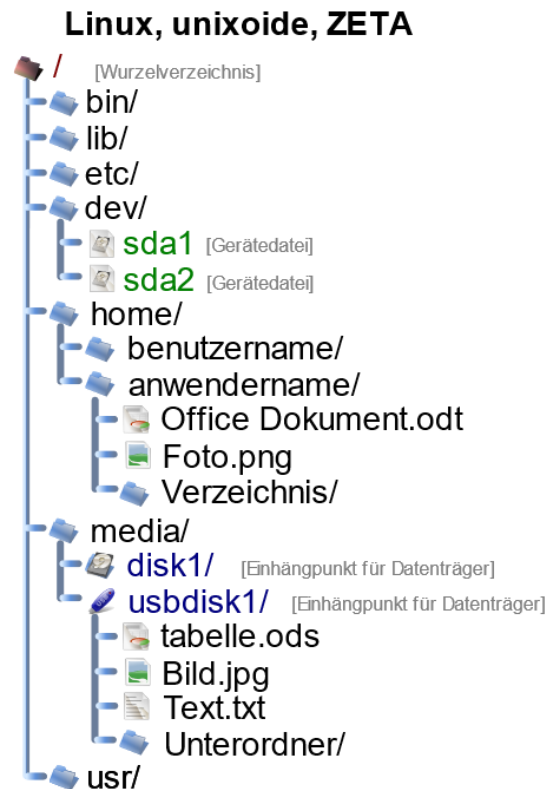
- Todas as **árvores orientadas** são **grafos orientados** !
- **MAS**, nem todos os grafos orientados são árvores orientadas !!
- **Árvore orientada:**
  - O nó **raiz** não tem qualquer arco incidente
  - Cada um dos outros nós tem **um só arco incidente**
  - Existe **um só caminho orientado** entre a **raiz** e cada um dos outros **nós**

# Exemplos de aplicação

- Árvores **genealógicas**
- Árvores de **pedigree**
- Torneios
- Hierarquia de uma organização
- Estrutura de um livro
- Taxonomias hierárquicas
- ...



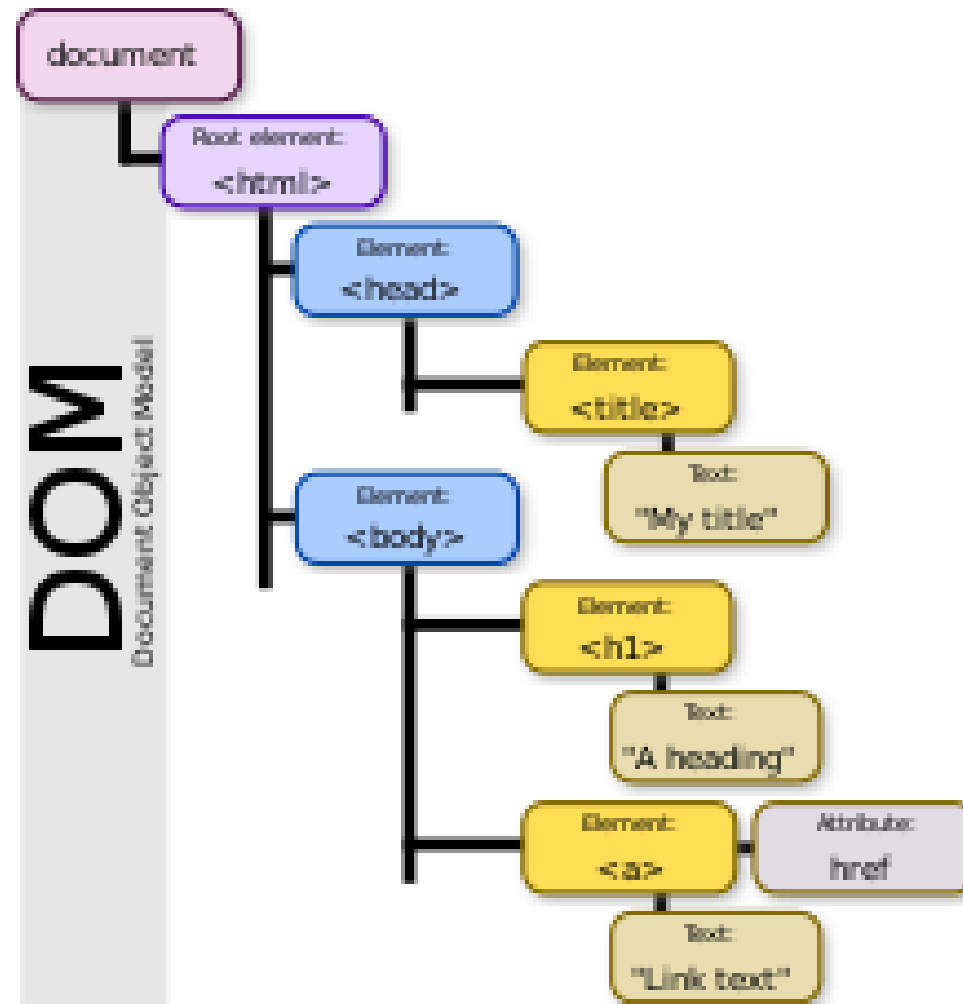
# Organização do sistema de ficheiros



[Wikipedia]



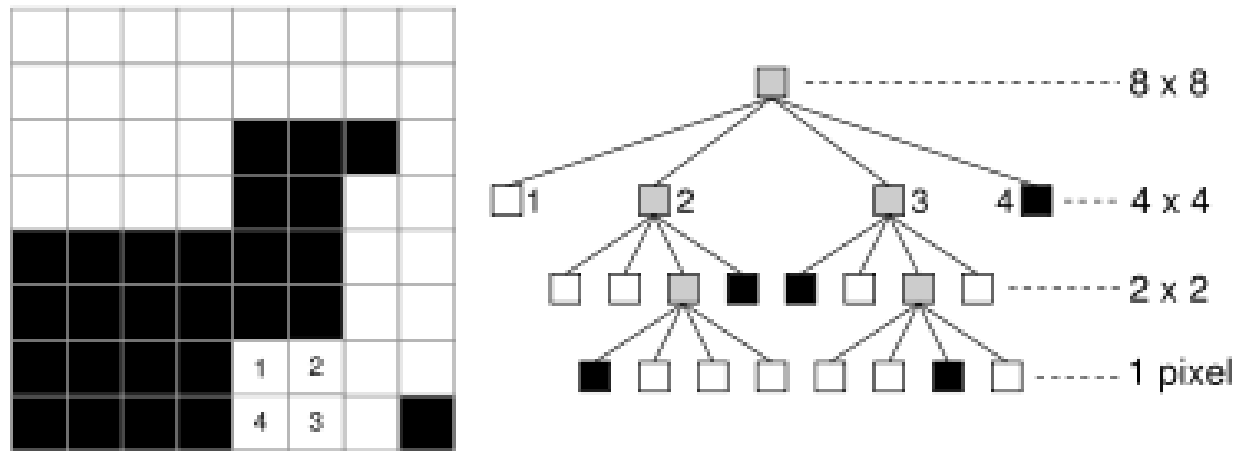
# DOM tree



[Wikipedia]

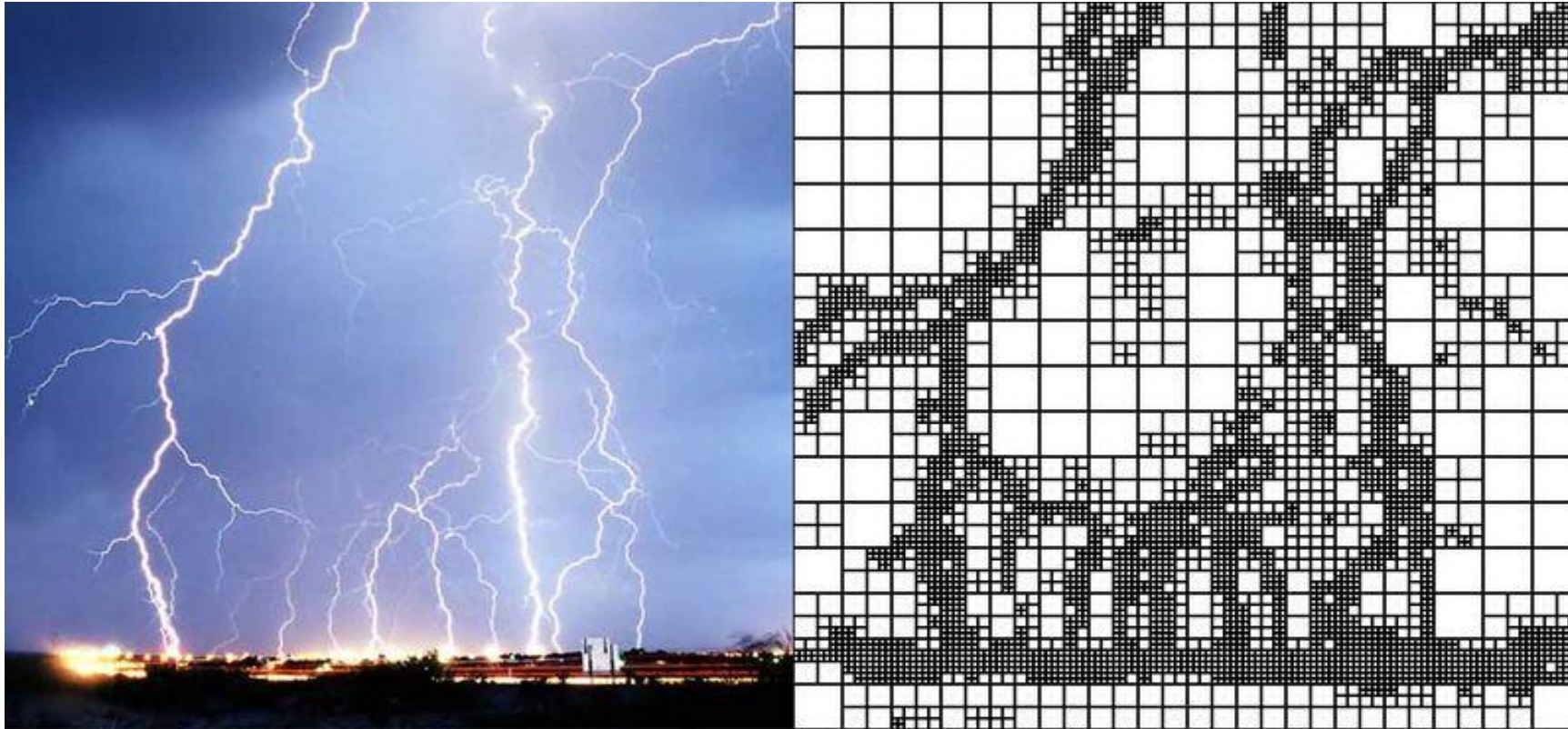
# Quadrees – Árvores quaternárias

- Representação de imagens binárias
  - Subdivisão recursiva em 4 quadrantes
  - Nós pretos, brancos e cinzentos



[Wikipedia]

# Quadtrees



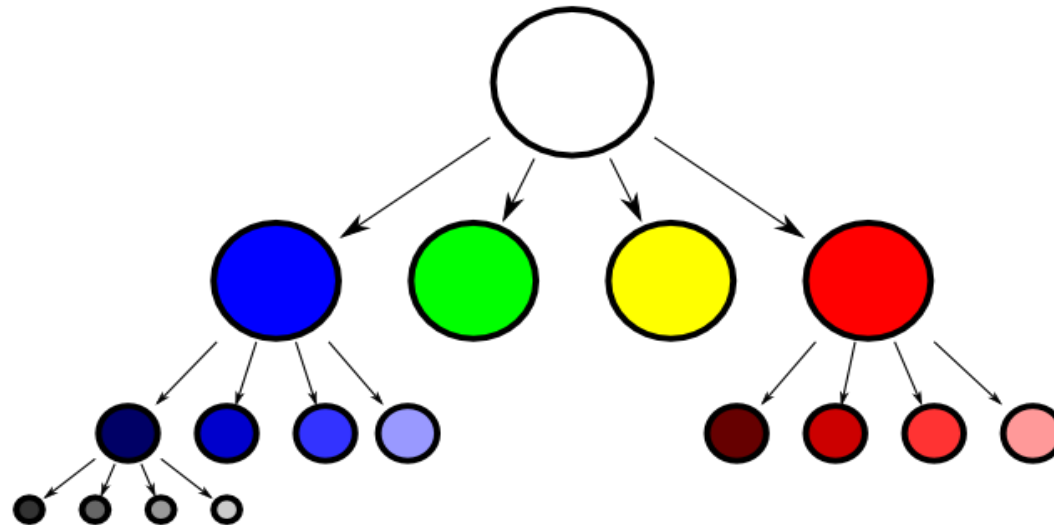
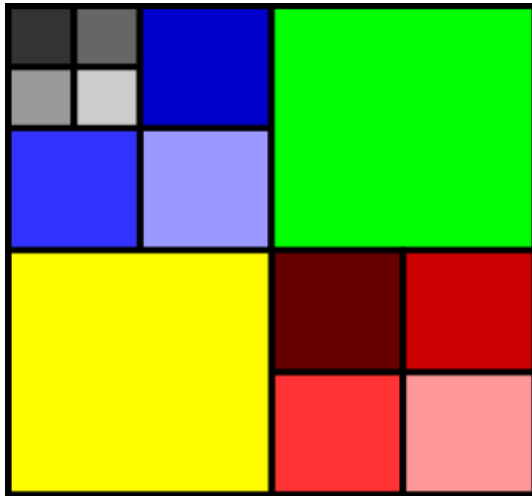
Originalbild

Aus Konturwerten  
erstellter Quadtree  
(Größe: 1229 Byte)

[Wikipedia]

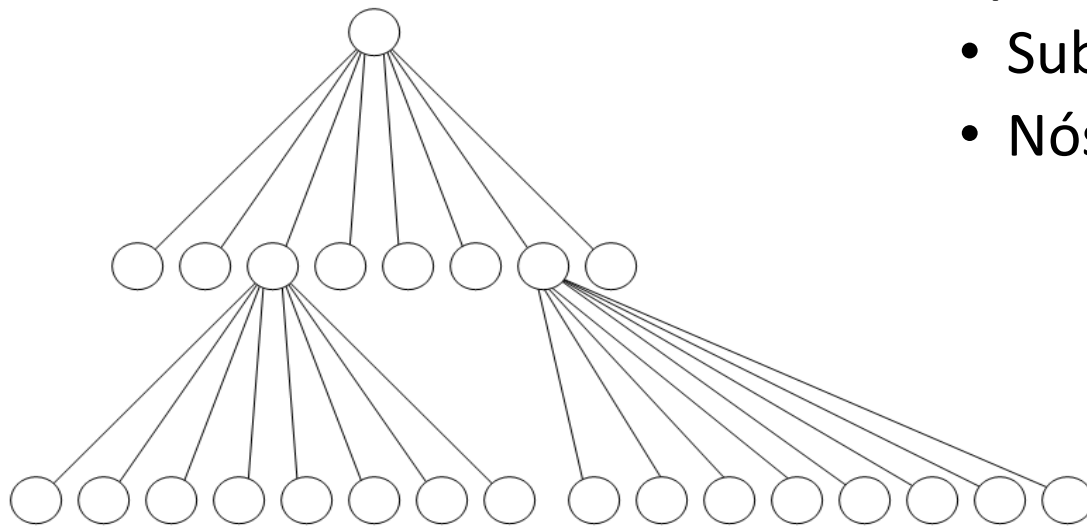
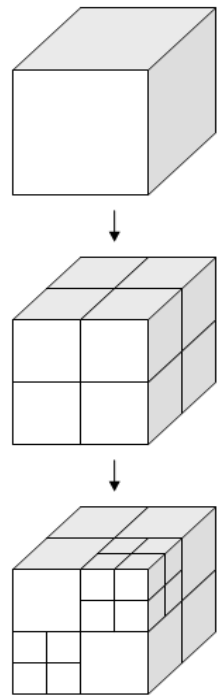
# Quadrees

- Representação de imagens a cores



[Wikipedia]

# Octrees – Árvores octais



- Representação de volumes
  - Subdivisão recursiva em 8 octantes
  - Nós pretos, brancos e cinzentos

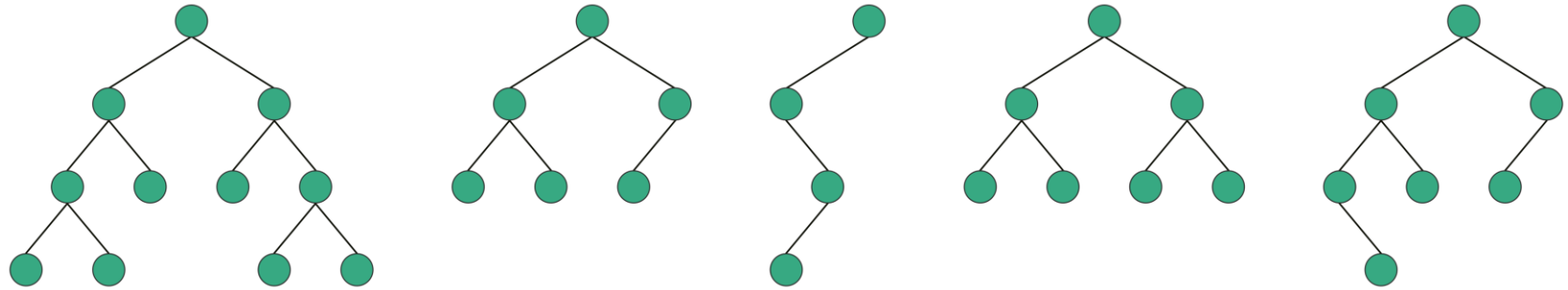
[Wikipedia]

# Mais exemplos de aplicação

- Herança **simples** em POO
- Comportamento dinâmico de algoritmos “**divide-and-conquer**”
- ...

# Resumo – Tipos de árvores

- Árvores **orientadas** vs não orientadas
- Árvores **binárias**, ternárias, quaternárias, ... , **m-árias**
- Árvores binárias **ordenadas** – Como ?
- Árvores binárias **equilibradas** – Como ?



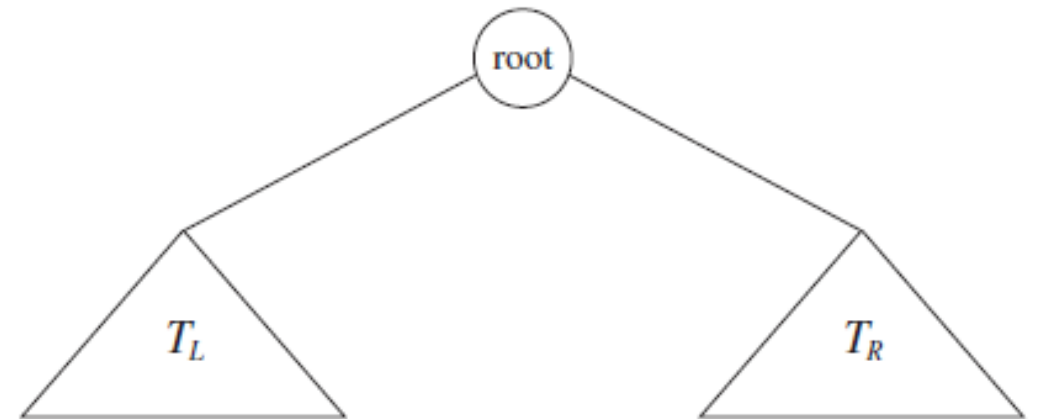
[towardsdatascience.com]

# Árvores Binárias



# Definição recursiva

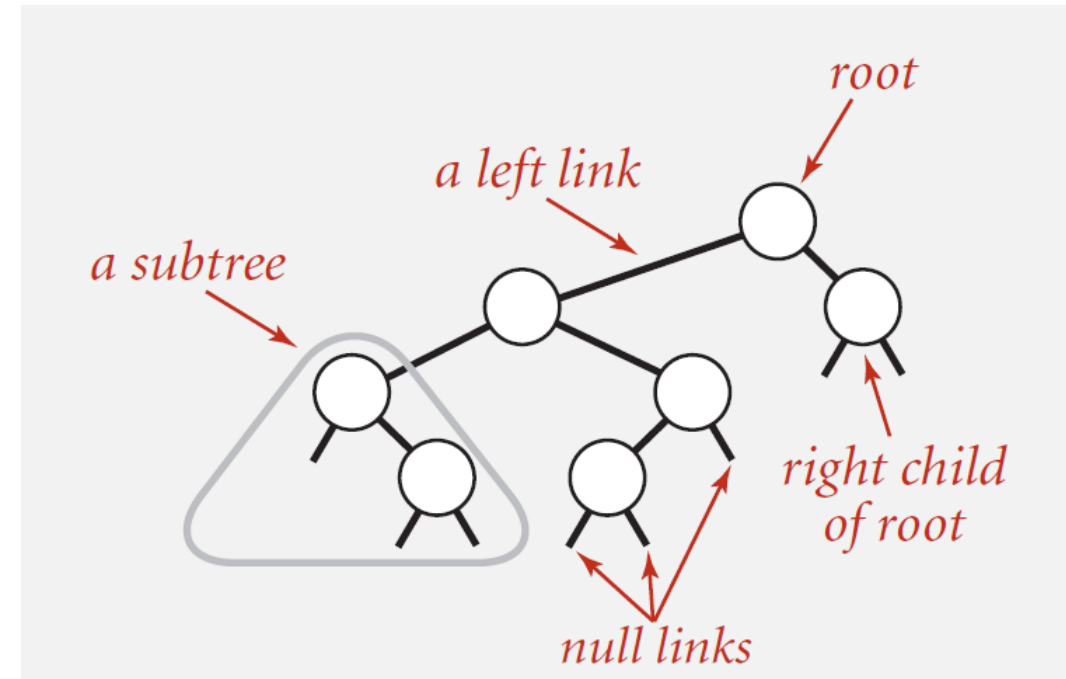
- Uma árvore binária é formada por um conjunto finito de nós ( $n \geq 0$ )
- Uma árvore binária é **vazia**
- **OU** é constituída por um nó **raiz** que referencia **duas (sub-)árvores** binárias disjuntas (**SAEsq** e **SADir**)
  - Arcos orientados para a SAEsq e para a SADir



[Weiss]

# Terminologia

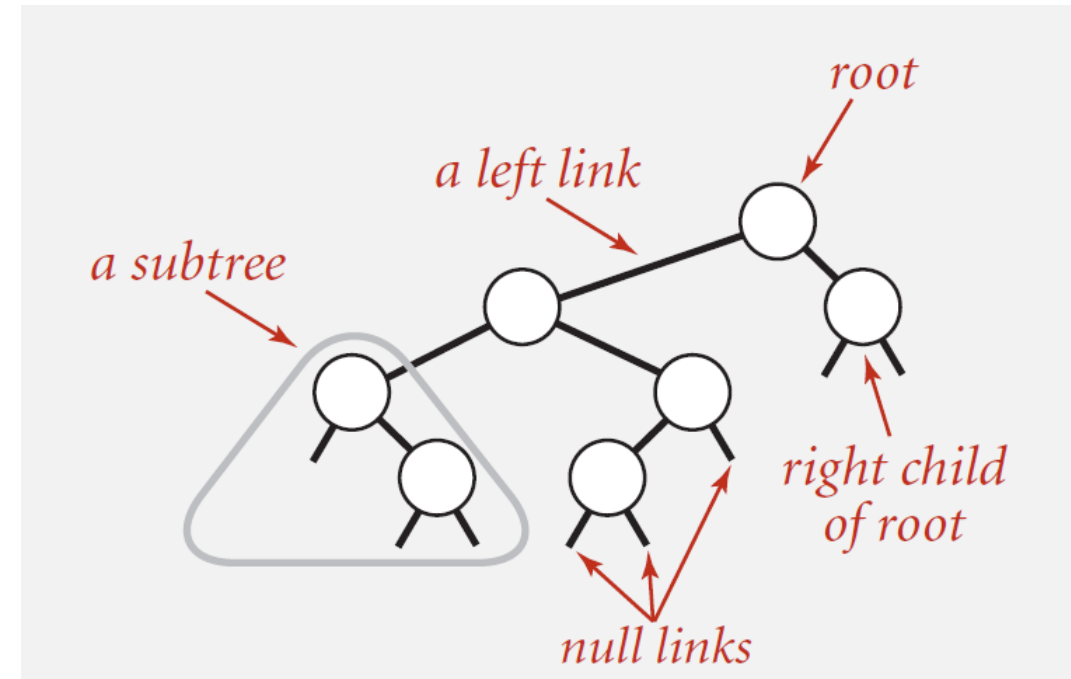
- **Grau** de um **nó**: nº das suas subárvores não-vazias
- **Grau** de uma **árvore** ?
- Nós **não-terminais** vs **Folhas**
- Pai, **filhos**, irmãos
- Antepassados, descendentes



[Sedgewick & Wayne]

# Terminologia

- **Nível** de um nó ?
- A **raiz** está no nível **0**
- Qual é a **definição recursiva** ?
- **Altura** de uma árvore ?
- Nº de arcos do **caminho mais longo** da raiz da árvore para uma das suas folhas
- Índice do **último nível**



[Sedgewick & Wayne]

# Algumas propriedades das árvores binárias

- Questões simples !!
- Nº **máximo** de nós no **nível i** ?
- Nº **máximo** de nós numa árvore de **altura h** ? Quando ?
- Nº **mínimo** de nós numa árvore de **altura h** ? Quando ?

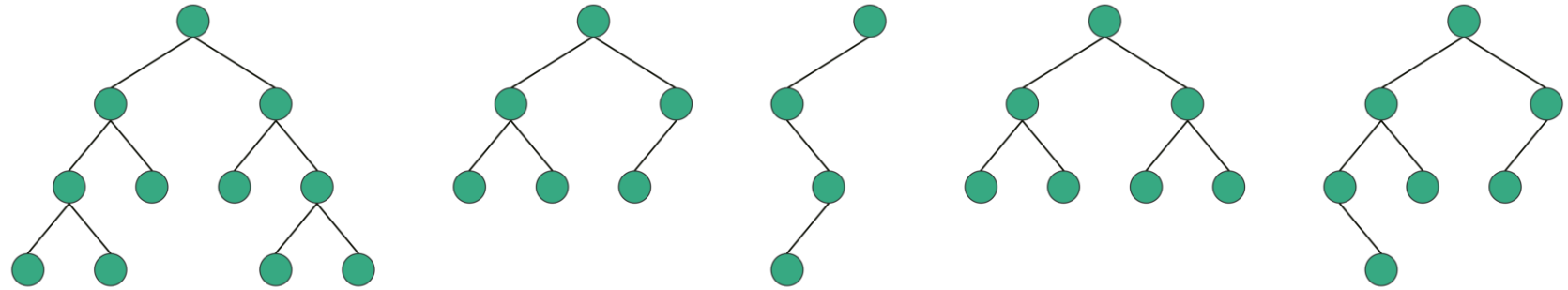
# Algumas propriedades das árvores binárias

- Questões simples !!
- Nº **máximo** de nós no **nível i** ?  $2^i$
- Nº **máximo** de nós numa árvore de **altura h** ? Quando ?  $2^{h+1} - 1$
- Nº **mínimo** de nós numa árvore de **altura h** ? Quando ?  $h + 1$

$$h + 1 \leq n \leq 2^{h+1} - 1$$

# Algumas propriedades das árvores binárias

- Completar a frase:
- A altura de uma árvore binária com  $n$  nós é pelo menos ... e quando muito ...
- Façam exemplos !!



[towardsdatascience.com]

# O TAD Árvore Binária

# TAD Árvore Binária – Funcionalidades

- Conjunto de **elementos** do **mesmo tipo**
- Armazenados **sem qualquer ordem particular**
- Procura / inserção / remoção / substituição
- Pertença
- **search() / insert() / remove() / replace()**
- **size() / isEmpty() / contains()**
- **create() / destroy()**





# Funcionalidades

```
#ifndef _INTEGERS_BINTREE_
#define _INTEGERS_BINTREE_

// JUST storing integers
typedef int ItemType;
typedef struct _TreeNode Tree;

Tree* TreeCreate(void);

void TreeDestroy(Tree** pRoot);
```



```
// Tree properties

int TreeIsEmpty(const Tree* root);

int TreeEquals(const Tree* root1, const Tree* root2);

int TreeMirrors(const Tree* root1, const Tree* root2);

// ...
```

# Funcionalidades

```
// Getters

int TreeGetNumberOfNodes(const Tree* root);

int TreeGetHeight(const Tree* root);

ItemType TreeGetMin(const Tree* root);

ItemType TreeGetMax(const Tree* root);

Tree* TreeGetPointerToMinNode(const Tree* root);

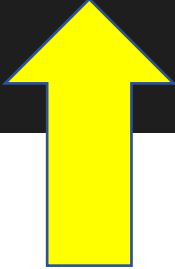
Tree* TreeGetPointerToMaxNode(const Tree* root);

// ...
```



# Funcionalidades

```
// Operations with items  
  
int TreeContains(const Tree* root, const ItemType item);  
  
int TreeAdd(Tree** pRoot, const ItemType item);  
  
int TreeRemove(Tree** pRoot, const ItemType item);  
  
// ...
```



# Funcionalidades

```
// Traversals
```

```
void TreeTraverseInPREOrder(Tree* root, void (*function)(ItemType* p));
```

```
void TreeTraverseINOrder(Tree* root, void (*function)(ItemType* p));
```

```
void TreeTraverseInPOSTOrder(Tree* root, void (*function)(ItemType* p));
```

```
// ...
```

# Ficheiro de teste

```
#include "IntegersBinTree.h"

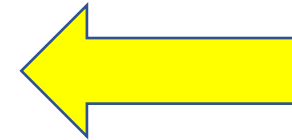
void printInteger(int* p) { printf("%d ", *p); }

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

int main(void) {
    Tree* tree = createExampleTree();

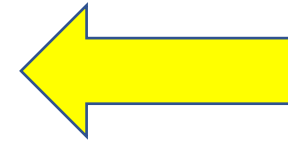
    printf("Created an example tree\n");

    if (TreeIsEmpty(tree)) {
        printf("The created tree is EMPTY\n");
    } else {
        printf("The created tree is OK\n");
    }
}
```

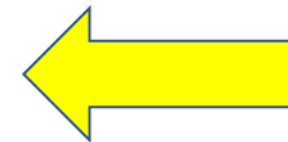


# Ficheiro de teste

```
printf("PRE-Order traversal : ");  
TreeTraverseInPREOrder(tree, printInteger);  
printf("\n");
```



```
printf("Multiply each value by 2\n");  
TreeTraverseInPREOrder(tree, multiplyIntegerBy2);  
printf("PRE-Order traversal : ");
```



# Possíveis representações internas

- **Array** – representação sequencial **por níveis** ✓
- **Lista de listas** – 1º filho e irmãos
- Nó com **2 ponteiros** para as subárvores esquerda e direita
- Nó com **mais 1 ponteiro** para o progenitor
- ...
- Que **operações** são mais **fáceis** / **difíceis** com uma dada representação interna?

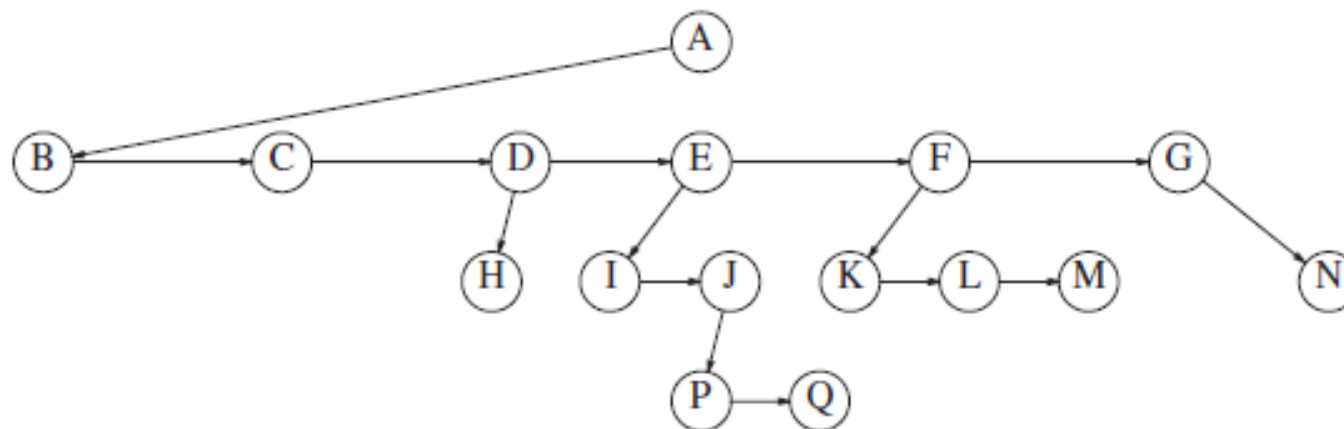
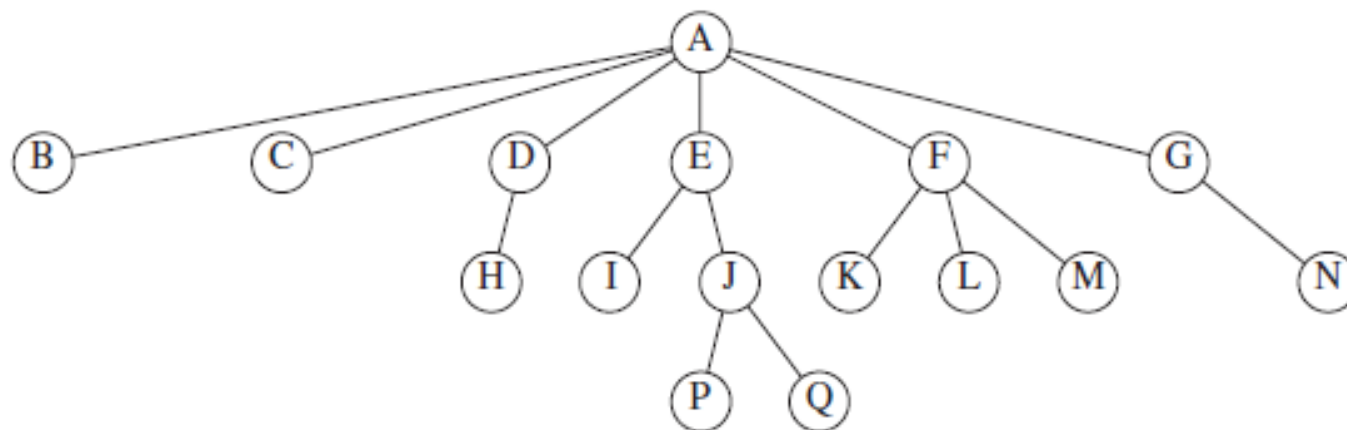
# Possíveis representações internas

- Não alterar as funcionalidades do TAD !!
- Que operações são mais fáceis / difíceis com uma dada representação interna?
- P.ex., listar por níveis
- Ou encontrar antepassados de um dado nó



# Lista de listas – 1º filho e irmãos

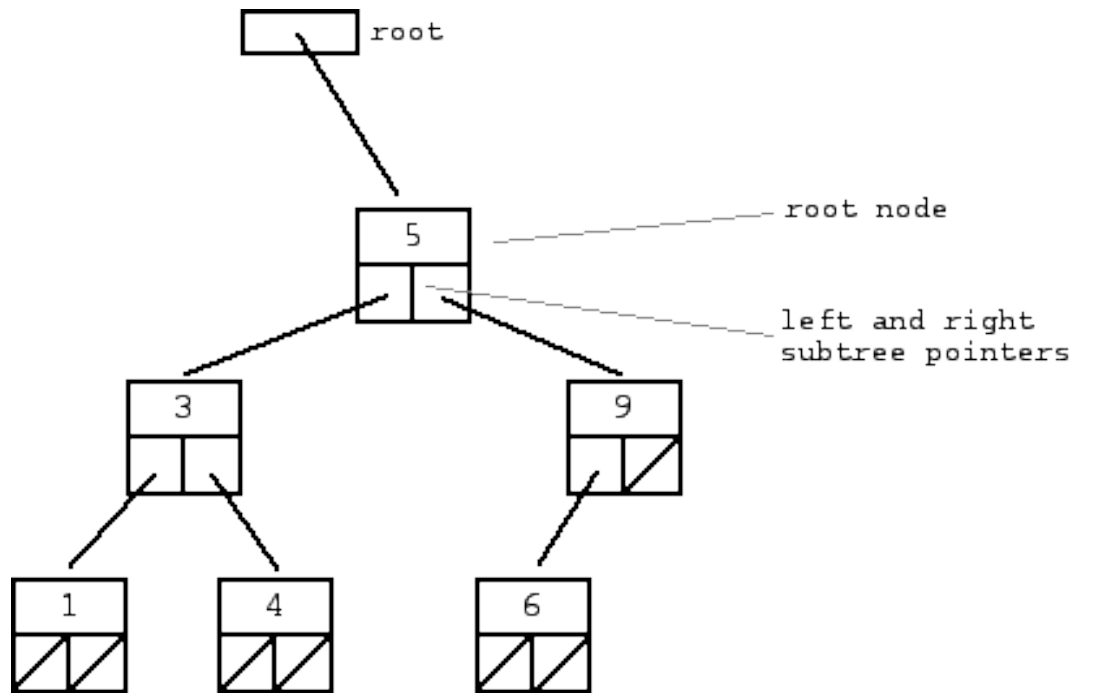
[Weiss]



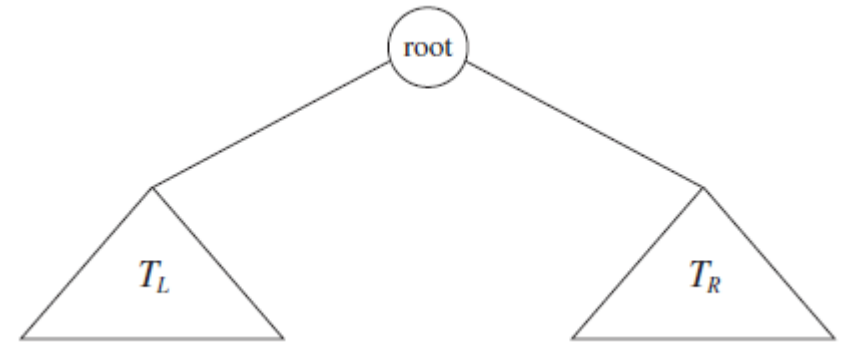
```
struct TreeNode
{
    Object    element;
    TreeNode *firstChild;
    TreeNode *nextSibling;
};
```

# Nó com 2 ponteiros

```
struct _TreeNode {  
    ItemType item;  
    struct _TreeNode* left;  
    struct _TreeNode* right;  
};
```



[stanford.edu]



[Weiss]

# Algoritmos Recursivos

## – Exemplos simples


# Desenvolver funções recursivas

- Contar o nº de nós de uma árvore
- Determinar a altura de uma árvore
- Destruir uma árvore
- Verificar se duas árvores são iguais
- As operações anteriores dependem da ordem de visita das subárvores ?

# Contar o nº de nós de uma árvore


```
int TreeGetNumberOfNodes(const Tree* root) {  
    if (root == NULL) return 0;  
  
    return 1 + TreeGetNumberOfNodes(root->left) +  
           TreeGetNumberOfNodes(root->right);  
}
```

# Determinar a altura de uma árvore



```
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;  
}
```

# Destruir uma árvore

```
void TreeDestroy(Tree** pRoot) {  
    Tree* root = *pRoot;  
  
    if (root == NULL) return;  
  
    TreeDestroy(&(root->left));  
  
    TreeDestroy(&(root->right));  
  
    free(root);  
  
    *pRoot = NULL;  
}
```



# Verificar se duas árvores são iguais

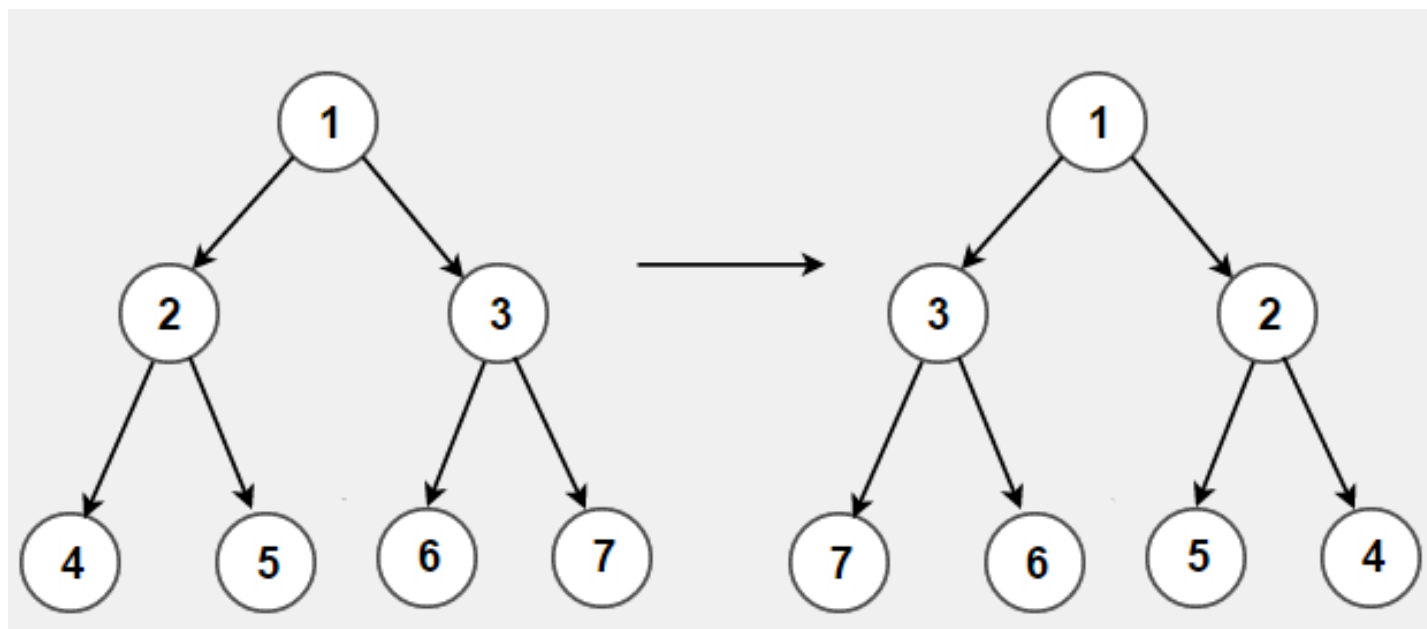


```
int TreeEquals(const Tree* root1, const Tree* root2) {  
    if (root1 == NULL && root2 == NULL) {  
        return 1;  
    }  
    if (root1 == NULL || root2 == NULL) {  
        return 0;  
    }  
    if (root1->item != root2->item) {  
        return 0;  
    }  
    return TreeEquals(root1->left,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



# Tarefa: duas árvores são espelhadas ?

- Desenvolver uma função recursiva



[techiedelight.com]

Tarefa: verificar se um item **pertence** à árvore

- Desenvolver uma função recursiva

# Tarefas adicionais – Funções recursivas

- Determinar o **valor** do **menor** elemento
- Determinar o **valor** do **maior** elemento
- Devolver um **ponteiro** para o nó contendo o **menor** elemento
- Devolver um **ponteiro** para o nó contendo o **maior** elemento