

Árvores

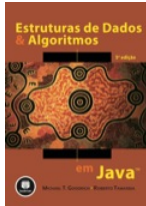
Roland Teodorowitsch

Algoritmos e Estruturas de Dados I - Escola Politécnica - PUCRS

3 de outubro de 2023

Introdução

Leitura(s) Recomendada(s)



Capítulo 7, Seções 7.1 e 7.2

GOODRICH, Michael T.; TAMASSIA, Roberto. **Estruturas de dados e algoritmos em Java**. Tradução: Bernardo Copstein. 5. ed. Porto Alegre: Bookman, 2013. xxii, 713 p. E-book. ISBN 9788582600191. Tradução de: Data Structures and Algorithms in Java, 5th Edition. Disponível em: <<https://integrada.minhabiblioteca.com.br/#/books/9788582600191/>>. Acesso em: 01 ago. 2023.

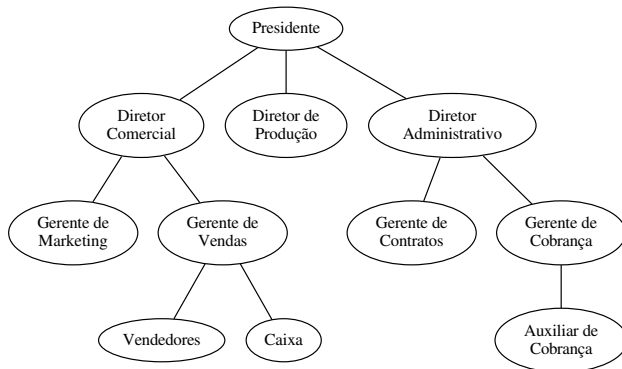
Conceitos e Terminologia

Árvores

- Estruturas de dados não lineares
- Permitem a implementação de vários algoritmos mais rápidos do que no uso de estruturas de dados lineares como as listas
- Fornecem uma forma natural de organizar os dados
 - Sistemas de arquivos
 - Bancos de dados
 - Sites da Web

Árvore

- Tipo abstrato de dados que armazena elementos de maneira **hierárquica**
- Normalmente, são desenhadas colocando-se os elementos dentro de elipses ou retângulos e conectando-os com linhas retas

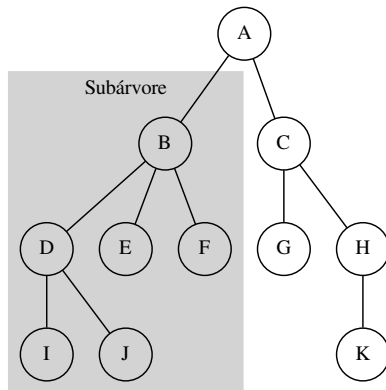


Definição

- Formalmente uma árvore **T** é definida como um conjunto finito de um ou mais **nodos**, com a seguinte propriedade:
 - Se **T** não é vazia, existe um nodo denominado **raiz** da árvore
 - Os demais nodos formam $m > 0$ conjuntos disjuntos S_1, S_2, \dots, S_m , sendo cada um destes conjuntos uma árvore
 - As árvores S_i ($1 \leq i \leq m$) são chamadas de **subárvores**
- Pela definição
 - Cada nodo da árvore é a raiz de uma subárvore

Definição

- Portanto, uma árvore pode ser representada da seguinte forma:



- A** é a raiz da árvore

Relacionamentos entre nodos

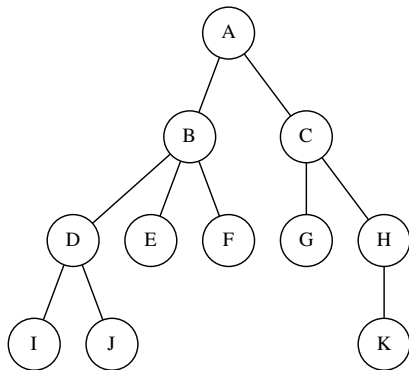
- Outra propriedade de uma árvore **T**:
 - Cada nodo **v** de **T** diferente da raiz tem um único nodo **pai**, **w**
 - Todo nodo com **pai w** é **filho** de **w**
- Pela definição
 - Uma árvore pode ser vazia, isto é, não possui nodos
 - Esta convenção permite que se defina uma árvore recursivamente
 - Uma árvore **T** ou está vazia, ou consiste de um nodo **r**, chamado de raiz de **T**, e um conjunto de árvores cujas raízes são filhas de **r**

Relacionamentos entre nodos

- Outros relacionamentos entre nodos
 - Dois nodos que são filhos de um mesmo pai são **irmãos**
 - Um nodo **v** é **externo** se **v** não tem filhos
 - Um nodo **v** é **interno** se tem um ou mais filhos
- Nodo **interno** também é conhecido como **galho**
- Nodo **externo** também é conhecido como **folha**

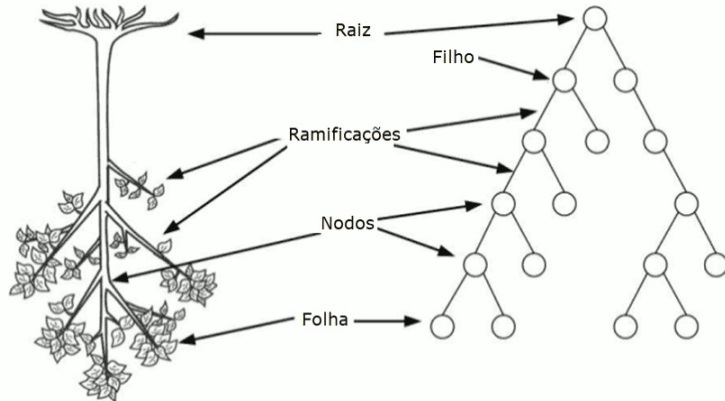
Relacionamentos entre nodos

- A raiz de uma árvore é chamada de **pai** de suas subárvores
- As raízes das subárvores de um nodo são chamadas de **irmãos**, que, por sua vez, são **filhos** de seu nodo pai



- **A** é pai de **B** e **C**
- **D**, **E** e **F** são irmãos
- **I** é filho de **J**

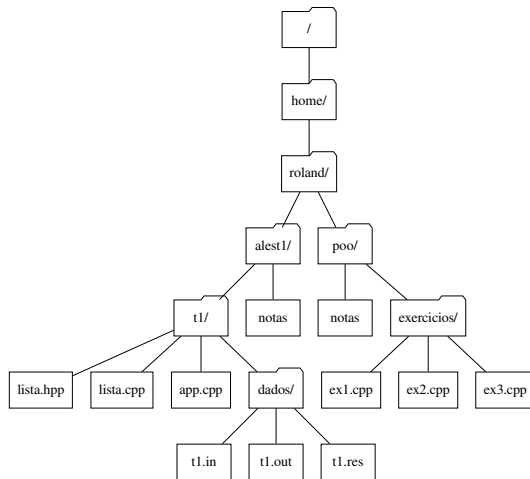
“Árvore Invertida”



Fonte: https://di.ubi.pt/~cbarrico/Disciplinas/AlgoritmosEstruturasDadosLEI/Downloads/Teorica_ConceitosGeraisArvores.pdf

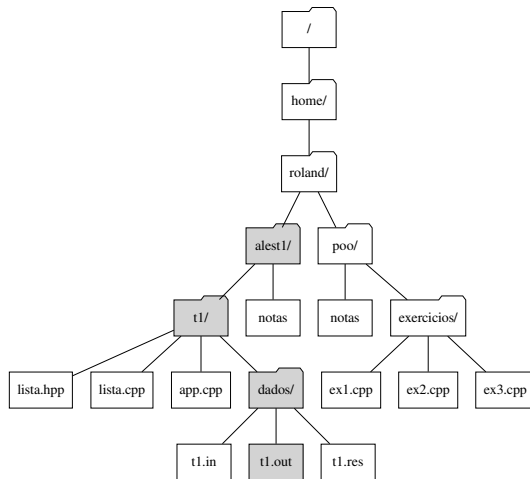
Exemplo

- Organização hierárquica dos arquivos nos sistemas operacionais é uma árvore
- Nós internos, neste caso, são associados a diretórios, e nós externos a arquivos
- No Linux o diretório raiz é “/”



Arestas e Caminhos em Árvores

- Uma aresta de uma árvore **T** é um par de nodos (**u,v**) tal que **u** é pai de **v**, ou vice-versa
- Um caminho de **T** é uma sequência de nodos tais que quaisquer dois nodos consecutivos da sequência formam uma aresta
- Exemplo: alest1/, t1/, dados/ e t1.out formam um caminho



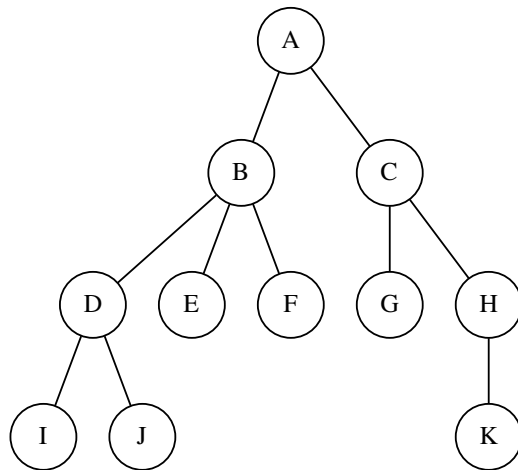
Grau, Nível e Altura

- Grau
 - É o número de subárvores de um nodo
 - Quando o grau é zero, ou seja, o nodo não possui filhos, ele é **folha**
- Nível de um nodo
 - É o número de linhas que liga o nodo à raiz, sabendo que a raiz é o nível zero
- Altura
 - É definida como sendo o nível mais alto da árvore

Grau, Nível e Altura

- Grau:

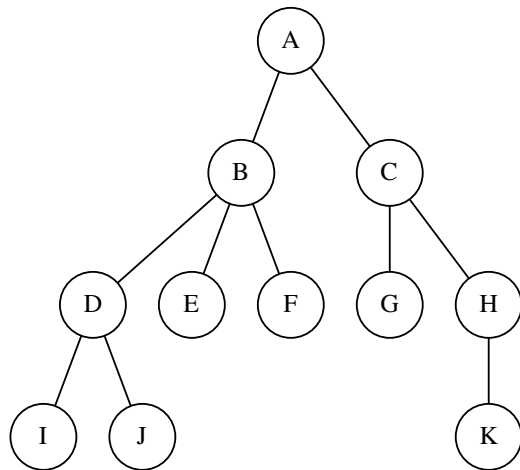
- Nodo A:
- Nodo B:
- Nodo C:
- Nodo D:
- Nodo E:
- Nodo F:
- Nodo G:
- Nodo H:
- Nodo I:
- Nodo J:
- Nodo K:



Grau, Nível e Altura

- Grau:

- Nodo A: 2
- Nodo B: 3
- Nodo C: 2
- Nodo D: 2
- Nodo E: 0
- Nodo F: 0
- Nodo G: 0
- Nodo H: 1
- Nodo I: 0
- Nodo J: 0
- Nodo K: 0



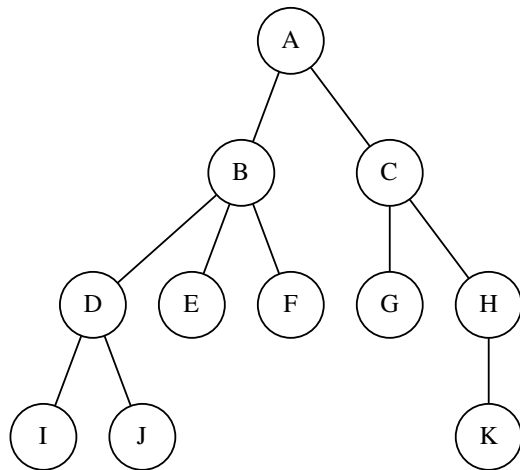
Grau, Nível e Altura

- Nível:

- Nodo A:
- Nodo B:
- Nodo C:
- Nodo D:
- Nodo E:
- Nodo F:
- Nodo G:
- Nodo H:
- Nodo I:
- Nodo J:
- Nodo K:

- Altura:

- Árvore:



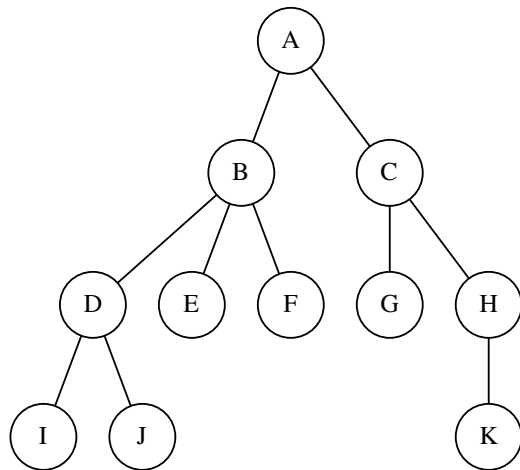
Grau, Nível e Altura

- Nível:

- Nodo A: 0
- Nodo B: 1
- Nodo C: 1
- Nodo D: 2
- Nodo E: 2
- Nodo F: 2
- Nodo G: 2
- Nodo H: 2
- Nodo I: 3
- Nodo J: 3
- Nodo K: 3

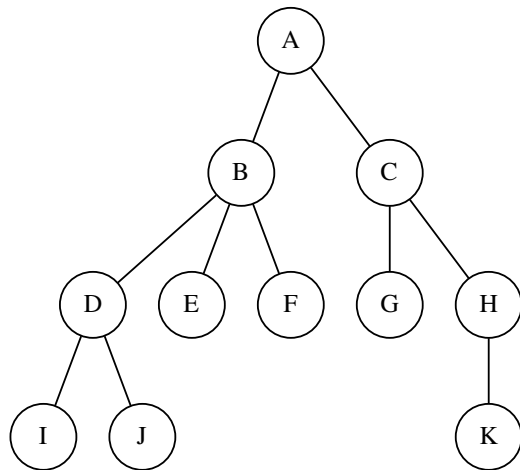
- Altura:

- Árvore: 3



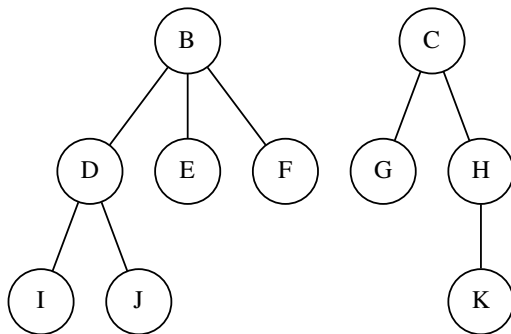
Grau, Nível e Altura

- Raiz: A
- Nodos internos: B, C, D, H
- Folhas: I, J, E, F, G, K



Floresta

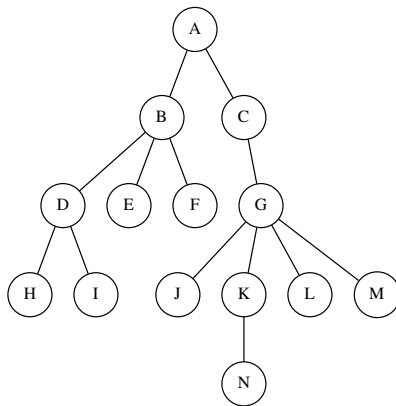
- Conjunto de uma ou mais árvores disjuntas
- Se eliminarmos o nodo raiz de uma árvore, obtém-se uma floresta
- Os filhos da raiz original irão se transformar nas raízes das novas árvores



Exercícios

Exercício 1

1 Analise a árvore abaixo.

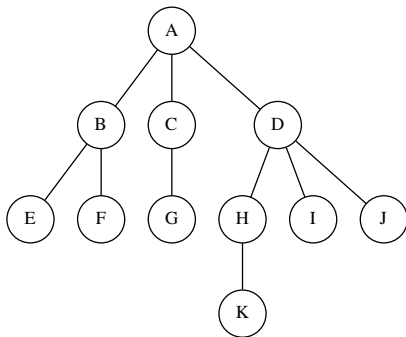


E responda:

- Qual é a altura da árvore?
- Quais são as folhas da árvore?
- Quais são os nodos irmãos?
- Os nodos D e G são pais de que nodos?
- Qual é o grau do nodo B?
- Qual é o grau do nodo G?
- Quais são os níveis dos nodos B, G, H, L e N?

Exercício 2

2 Analise a árvore abaixo.



E responda:

- Qual é a altura da árvore?
- Quais são as folhas da árvore?
- Quais são os nodos irmãos?
- Quais são os nodos internos?
- Qual é o grau do nodo C?
- Qual é o grau do nodo D?
- Quais são os níveis dos nodos C, H e K?

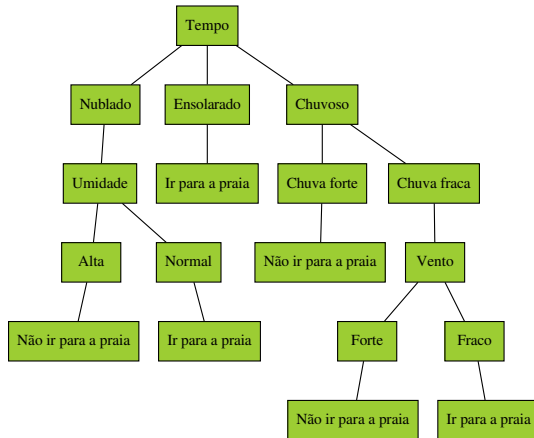
Aplicações

Aplicações

- Árvores são estruturas de dados adequadas para representar diversos tipos de informações
- Várias aplicações que podem utilizar árvores

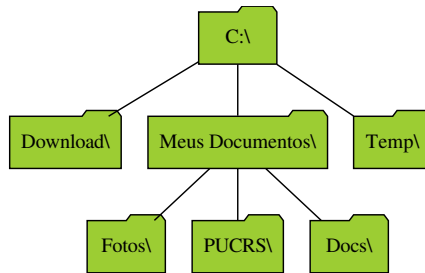
Árvore de decisão

- Exemplo: Ir ou não ir para praia?



Representação de estruturas/informações hierárquicas

- Árvore genealógica
- Organização de livros e documentos
- Organização hierárquica de cargos de uma empresa
- Sistemas de arquivos



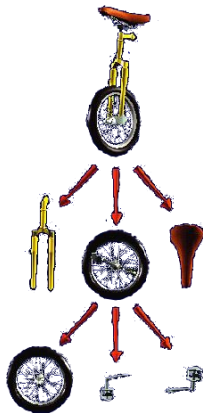
Representação de estruturas/informações hierárquicas

- Arquivos HTML e XML

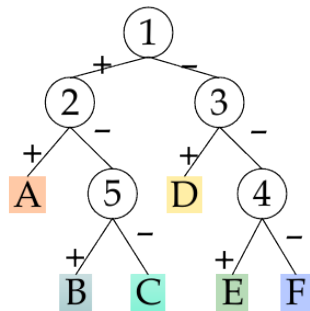
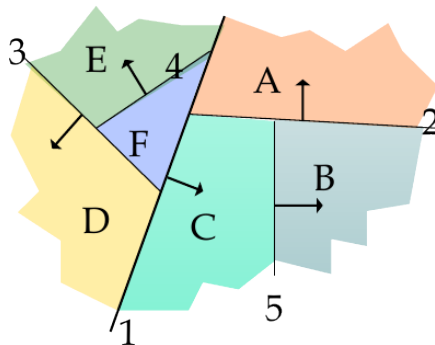
```
<?xml version="1.0"?>
<Company>
  <Employee>
    <FirstName>Maria</FirstName>
    <LastName>Silva</LastName>
    <PhoneNumber>(51)98986767</PhoneNumber>
    <Email>maria.silva@gmail.com</Email>
    <Address>
      <Street>Rua dos Andradas 1586/202</Street>
      <City>Porto Alegre</City>
      <State>RS</State>
      <Zip>90021-212</Zip>
    </Address>
  </Employee>
</Company>
```

Computação Gráfica

- Grafo de cena

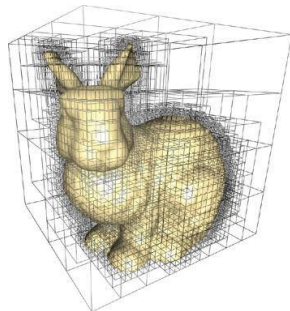
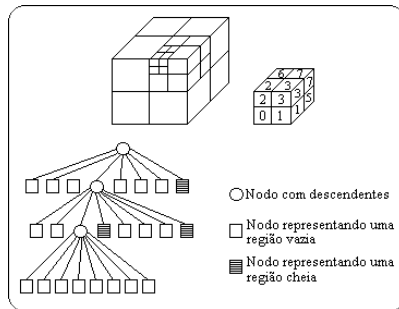
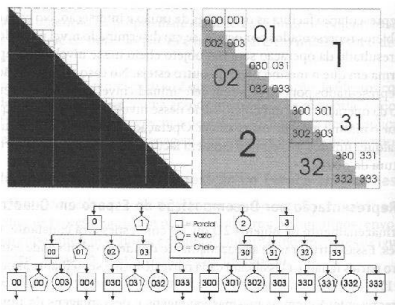


- Algoritmos para remoção de superfícies escondidas (árvore *Binary Space-Partitioning*)



Computação Gráfica

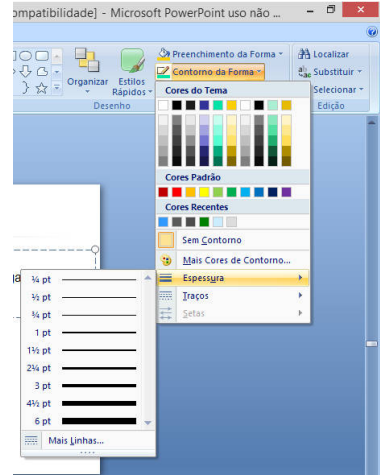
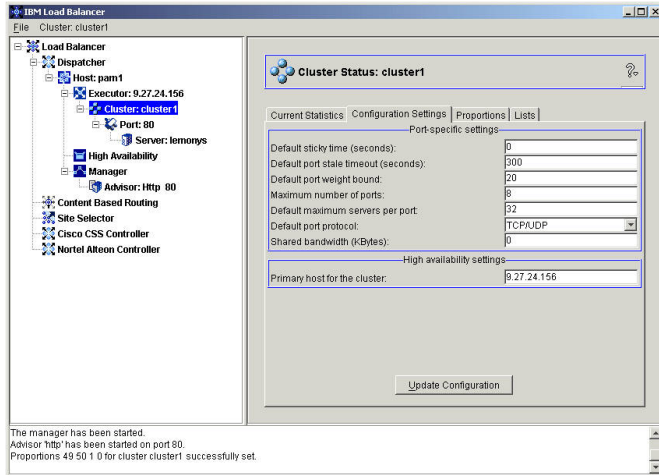
- Representação de objetos (Quadtree e Octree), etc.



Eliminatórias de Campeonatos

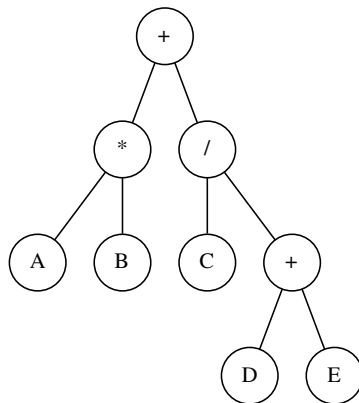


Interfaces Gráficas com o Usuário

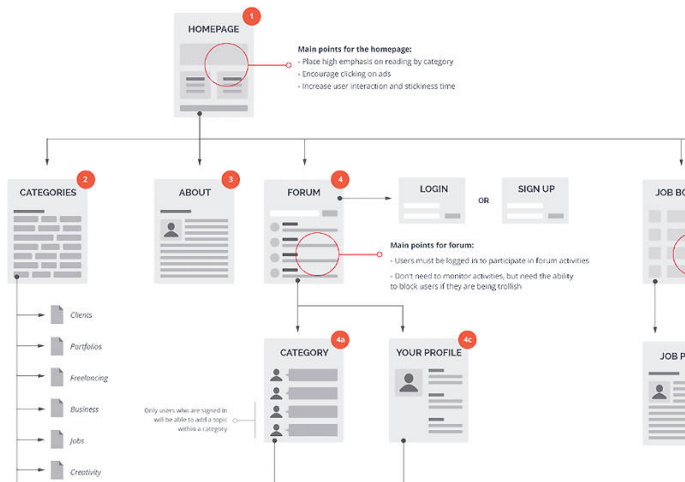


Expressões Aritméticas

- Pode-se usar árvores para representar e avaliar expressões aritméticas
- Exemplo: $A * B + C / (D + E)$



Organização das Páginas de um Site



Fonte: <https://dribbble.com/shots/1198252-Sitemap-For-Student-Guide>

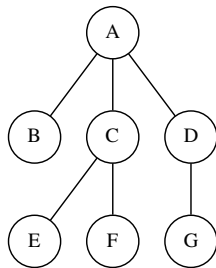
Representação na Memória

Representação na Memória

- Da mesma forma que as estruturas de dados lineares, podemos alocar as árvores de duas maneiras
 - Por contiguidade
 - Por encadeamento

Representação por Contiguidade

- A árvore é armazenada em um arranjo
- Cada posição por arranjo pode, por exemplo, conter, além da informação do nodo, referências aos nodos filhos



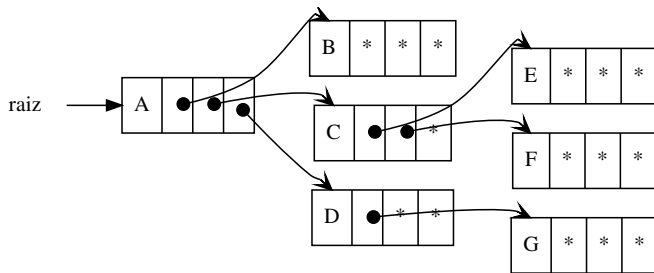
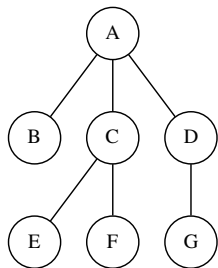
índice	conteúdo	filho1	filho2	filho3
0	A	1	2	3
1	B	-1	-1	-1
2	C	4	5	-1
3	D	6	-1	-1
4	E	-1	-1	-1
5	F	-1	-1	-1
6	G	-1	-1	-1

Representação por Contiguidade

- Vantagem:
 - Forma de armazenar árvores em arquivos
- Desvantagem:
 - Quantidade de processamento para inserção, remoção ou mesmo localização de um nodo

Representação por Encadeamento

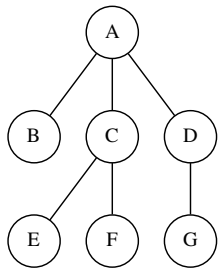
- Forma **mais usual**
- Facilita as operações de inserção, remoção e pesquisa na árvore
- Cada nodo conterá, além da informação, as referências das suas subárvores



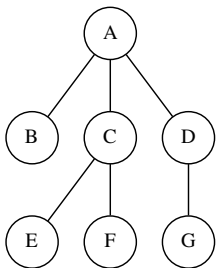
Representação por Encadeamento

- No caso de árvores “genéricas”, cada nodo pode ter uma quantidade de subárvores diferentes
- Torna-se necessário:
 - Limitar, ou seja, determinar o número máximo de subárvores que cada nodo deve conter
 - Ou ter uma lista de subárvores
- Isso é necessário, pois os nodos de uma mesma árvore são todos do mesmo tipo

Exemplo



Exemplo



```

#include <iostream>

using namespace std;

struct Node {
    char info; Node *child1, *child2, *child3;
    Node(char i, Node *c1 = nullptr, Node *c2 = nullptr, Node *c3 = nullptr) {
        info = i; child1 = c1; child2 = c2; child3 = c3;
        cout << "+_Node("<< info << ")_criado..." << endl;
    }
    ~Node() { cout << "-_Node("<< info << ")_destruido..." << endl; }
};

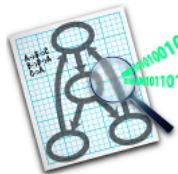
int main() {
    Node *b = new Node('B'), *e = new Node('E'),
    *f = new Node('F'), *g = new Node('G'); // Cria folhas
    Node *c = new Node('C',e,f), *d = new Node('D',g); // Cria intermediarios
    Node *root = new Node('A',b,c,d); // Cria raiz

    delete b; delete e; delete f; delete g; // Desaloca folhas
    delete c; delete d; // Desaloca intermediarios
    delete root; // Desaloca raiz

    return 0;
}

```

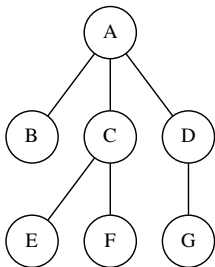
GraphViz



- Graphviz (abreviação de *Graph Visualization Software*) é uma ferramenta de código-fonte aberto para desenhar grafos (formados por nodos e arestas) especificados a partir de uma linguagem de escrita chamada DOT (que usa a extensão “gv”)
- GraphViz também provê bibliotecas para que aplicações possam usar suas facilidades
- É um *software* livre licenciado através da Licença Pública Eclipse
- Página: <https://graphviz.org/>
- Há várias opções para gerar imagens a partir dos arquivos no formato DOT
 - <https://dreampuf.github.io/GraphvizOnline/>

GraphViz

Grafo:



Versão 1:

```
graph "Árvore_A" {
  node [shape=circle]
  A -- { B C D }
  C -- { E F }
  D -- G
}
```

Versão 2:

```
graph "Árvore_A(Versão2)" {
  node [shape=circle]
  A -- B
  A -- C
  A -- D
  C -- E
  C -- F
  D -- G
}
```

Exercícios

Exercício 3

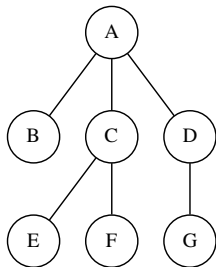
- 3 Considere a árvore e o programa que cria esta árvore correspondente, usando estruturas encadeadas em C++, ambos apresentados na próxima página. Observe que o nodo declarado (`struct Node`) armazena um caractere (`char`) e suporta até 3 subárvores (ou seja, cada nodo pode ter 3 nodos filhos). Para este programa, implemente as seguintes funções:
- `void clean(Node *root)`: que recebe o endereço de um nodo e faz a desalocação de todos os nodos da árvore a partir do nodo recebido (`root`) – esta função deve ser implementada de forma recursiva;
 - `string strGraphViz(Node *root)`: que recebe o endereço de um nodo e gera uma cadeia de caracteres (`string`) com representação da árvore no formato DOT, usado pelo GraphViz – esta função deve: imprimir a parte inicial do formato DOT; chamar um outro método recursivo (por exemplo, `string strNode(Node *node)`) para gerar a lista de arestas entre os nodos; imprimir a parte final do formato DOT.

Rode o seu programa, por exemplo, com:

```
./exercicio03 > exercicio03.gv
```

Use o site <https://dreampuf.github.io/GraphvizOnline/> para verificar se o arquivo `exercicio03.gv` foi corretamente gerado.

Exercício 3



```

#include <iostream>
#include <sstream>

using namespace std;

struct Node {
    char info; Node *child1, *child2, *child3;
    Node(char i, Node *c1 = nullptr, Node *c2 = nullptr, Node *c3 = nullptr) {
        info = i; child1 = c1; child2 = c2; child3 = c3;
        cerr << "+_Node("<< info << ")_criado..." << endl;
    }
    ~Node() { cerr << "-_Node("<< info << ")_destruido..." << endl; }
};

string strGraphViz(Node *root) { stringstream ss; /* IMPLEMENTE AQUI */
return ss.str(); }

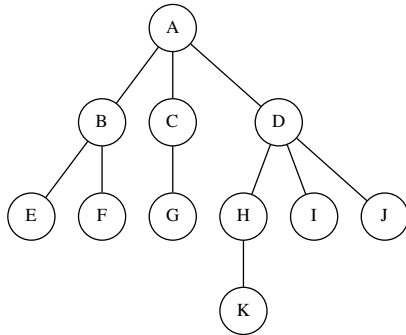
void clean(Node *root) { /* IMPLEMENTE AQUI */ }

int main() {
    Node *b = new Node('B'), *e = new Node('E'),
        *f = new Node('F'), *g = new Node('G'); // Folhas
    Node *c = new Node('C',e,f), *d = new Node('D',g); // Intermediarios
    Node *root = new Node('A',b,c,d); // Raiz
    strGraphViz(root);
    clean(root);
    return 0;
}

```


Exercício 4

- 4 Considere a árvore abaixo e, usando como modelo o código do exercício 3 (resolvido), implemente um programa em C++ que crie esta árvore em memória, que a imprima no formato DOT e que desaloque corretamente todos os nodos da árvore. Use o site <https://dreampuf.github.io/GraphvizOnline/> para verificar se o arquivo DOT foi corretamente gerado.



TAD para Árvore

TAD para Árvore

- Um TAD para árvore:
 - Armazena elementos em nodos
 - O posicionamento dos nodos satisfaz as relações pai-filho
 - Operações consideram as propriedades desta estrutura de dados hierárquica

TAD para Árvore

- Uma árvore deve disponibilizar métodos de acesso que retornam e aceitam posições:
 - `root()`: retorna a raiz da árvore
 - `parent(v)`: retorna o nodo pai de v , ocorrendo um erro se for a raiz
 - `children(v)`: retorna os filhos do nodo v

TAD para Árvore

- Métodos de consulta:
 - `isInternal(v)`: testa se um nodo v é interno e retorna `true` ou `false`
 - `isExternal(v)`: testa se um nodo v é externo e retorna `true` ou `false`
 - `isRoot(v)`: testa se um nodo v é raiz e retorna `true` ou `false`

TAD para Árvore

- Métodos “genéricos” (não estão necessariamente relacionados com sua estrutura):
 - `size()`: retorna o número de nodos na árvore
 - `isEmpty()`: testa se a árvore tem ou não tem algum nodo
 - `iterator()`: retorna um iterator de todos os elementos armazenados nos nodos da árvore
 - `positions()`: retorna uma coleção com todos os nodos da árvore
 - `replaceElement(v,e)`: retorna o elemento armazenado em `v` e o substitui por `e`

Árvore Binária

Exercícios

Exercício 10

- 10 Crie uma nova classe para um nodo de uma árvore binária que tenha a seguinte estrutura:
- Referência ao dado
 - Referência ao nodo da esquerda
 - Referência ao nodo da direita
 - Referência ao nodo pai

Créditos

Créditos

- Estas lâminas foram adaptadas do material desenvolvido pela professora Isabel Harb Manssour.

Soluções

Exercício 1

Qual é a altura da árvore?

4

Quais são as folhas da árvore?

H, I, E, F, J, N, L, M

Quais são os nodos irmãos?

B e C; D, E e F; H e I; J, K, L e M

Os nodos D e G são pais de que nodos?

D é pai de H e I; G é pai de J, K, L e M

Qual é o grau do nodo B?

3

Qual é o grau do nodo G?

4

Quais são os níveis dos nodos B, G, H, L e N?

B, nível 1; G, nível 2; H, nível 3; L, nível 3; N, nível 4

Exercício 2

Qual é a altura da árvore?

3

Quais são as folhas da árvore?

E, F, G, K, I, J

Quais são os nodos irmãos?

B, C e D; E e F; H, I e J

Quais são os nodos internos?

B, C, D e H

Qual é o grau do nodo C?

1

Qual é o grau do nodo D?

3

Quais são os níveis dos nodos C, H e K?

C, nível 1; H, nível 2; K, nível 3

Exercício 3: exercicio03-resp.cpp

```

string strNode(Node *node) {
    stringstream ss;
    if ( node->child1 != nullptr ) ss << "  " << node->info << "  --  " << node->child1->info << endl << strNode(node->child1);
    if ( node->child2 != nullptr ) ss << "  " << node->info << "  --  " << node->child2->info << endl << strNode(node->child2);
    if ( node->child3 != nullptr ) ss << "  " << node->info << "  --  " << node->child3->info << endl << strNode(node->child3);
    return ss.str();
}

string strGraphViz(Node *root) {
    stringstream ss;
    ss << "graph TD; " << endl << "  node([ " << endl << strNode(root) << " ] )" << endl;
    return ss.str();
}

void clean(Node *root) {
    if ( root->child1 != nullptr ) clean(root->child1);
    if ( root->child2 != nullptr ) clean(root->child2);
    if ( root->child3 != nullptr ) clean(root->child3);
    delete root;
}

```

Exercício 4: exercicio04.cpp

```

#include <iostream>
#include <sstream>

using namespace std;

struct Node {
    char info; Node *child1, *child2, *child3;
    Node(char i, Node *c1 = nullptr, Node *c2 = nullptr, Node *c3 = nullptr) {
        info = i; child1 = c1; child2 = c2; child3 = c3;
        cerr << "+_Node("<< info << ")_criado..." << endl;
    }
    ~Node() { cerr << "-_Node("<< info << ")_destruido..." << endl; }
};

string strNode(Node *node) {
    stringstream ss;
    if ( node->child1 != nullptr ) ss << "uu" << node->info << "u--u" << node->child1->info << endl << strNode(node->child1);
    if ( node->child2 != nullptr ) ss << "uu" << node->info << "u--u" << node->child2->info << endl << strNode(node->child2);
    if ( node->child3 != nullptr ) ss << "uu" << node->info << "u--u" << node->child3->info << endl << strNode(node->child3);
    return ss.str();
}

string strGraphViz(Node *root) {
    stringstream ss;
    ss << "graph TD; " << endl << "    node(( )) " << endl << strNode(root) << " " << endl;
    return ss.str();
}

void clean(Node *root) {
    if ( root->child1 != nullptr ) clean(root->child1);
    if ( root->child2 != nullptr ) clean(root->child2);
    if ( root->child3 != nullptr ) clean(root->child3);
    delete root;
}

int main() {
    Node *e = new Node('E'), *f = new Node('F'), *g = new Node('G'), *k = new Node('K'),
          *i = new Node('I'), *j = new Node('J'), *b = new Node('B',e,f), *c = new Node('C',g),
          *h = new Node('H',k), *d = new Node('D',h,i,j), *a = new Node('A',b,c,d);
    cout << strGraphViz(a); clean(a);
}

```

