Estruturas de Dados I

Árvores

Igor Machado Coelho

30/09/2020

Section 1

Árvores

Pré-Requisitos

São requisitos para essa aula:

- Introdução/Fundamentos de Programação (em alguma linguagem de programação)
- Interesse em aprender C/C++
- Noções de recursividade
- Noções de tipos de dados
- Noções de listas e encadeamento

Agradecimentos especiais ao prof. Fabiano Oliveira do IME/UERJ, cujo conteúdo didático forma a base desses slides

Section 2

Tipo Abstrato: Árvore

Árvore

A Árvore (do inglês *Tree*) é um Tipo Abstrato de Dado (TAD) que pode assumir duas formas:

- árvore T vazia, denotada por $T = \emptyset$
- árvore T composta por:
  - um nó R chamado de *nó raiz*
  - 0 ou mais árvores disjuntas T_1, T_2, ..., associadas a R; tais árvores são chamadas de *subárvores*

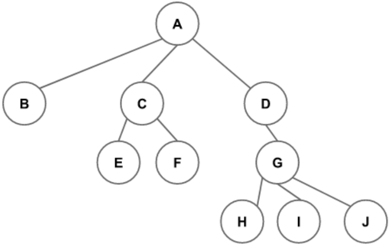


Figure 1: Representação de árvore

Nomenclatura

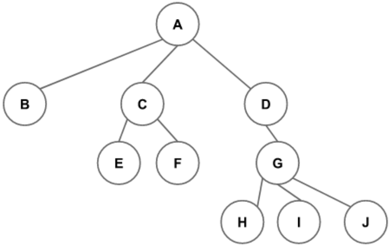
Um conjunto de árvores é chamado *floresta*

Se T é uma árvore com raiz R, então:

- os *nós de T* são todas as raízes de subárvores de R, além da raiz de T
- um nó com 0 filhos é chamado de *folha* (do inglês *leaf*)
- se um nó F é um *filho* de um nó P, denominamos P como *pai* de F
- a raiz é um nó *ancestral* de todos nós da árvore
- todos os nós da árvore são descendentes do nó raiz

Caminhos

Um caminho em uma árvore é uma sequência de nós com relação *filho de* ou *pai de*:

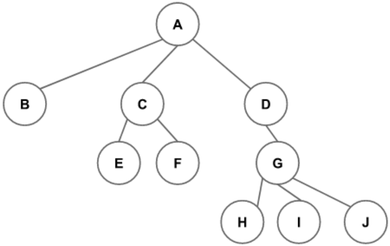


Exemplos:

- E,C,A
- D,G,I
- C,A,D,G

Tamanho de Caminhos e Níveis

O *tamanho de um caminho* consiste no número de nós. O *nível* de um nó é o tamanho de seu caminho até a raiz:

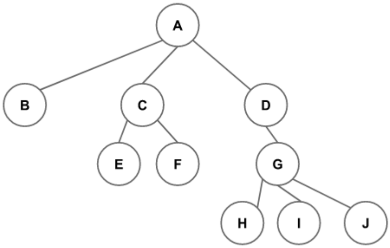


Nível de: A=1; C=2; F=3; H=4.

Desafio: em cursos de Teoria dos Grafos é provado que existe um único caminho conectando dois nós na árvore. *Utilize sua intuição para verificar esta afirmação!*

Alturas

A *altura* de nó X é o tamanho do maior caminho que conecta X a uma folha descendente. Denotamos a altura de X por $h(X)$:



Alturas: $h(B) = 1$; $h(C) = 2$; $h(D) = 3$; $h(A) = 4$.

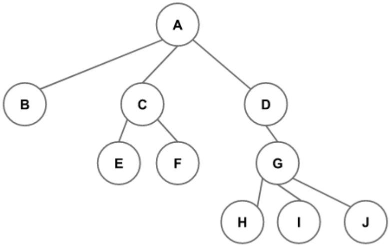
A *altura da árvore* é a altura de sua raiz!

No exemplo, $h(T) = h(A) = 4$.

Aridade

Uma árvore é dita *ordenada* se há uma ordem associada aos filhos de cada nó.

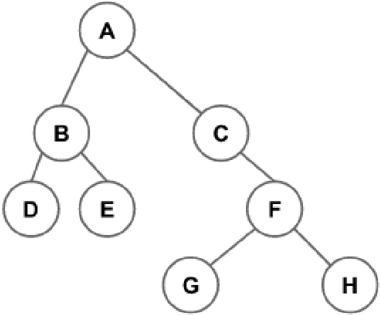
Uma árvore é dita *m-ária* se cada nó é *limitado a um máximo* de *m* filhos.



A árvore acima é ternária (podendo também ser 4-ária, 5-ária, 6-ária, . . .), mas *não é binária*!

Filho esquerdo e direito

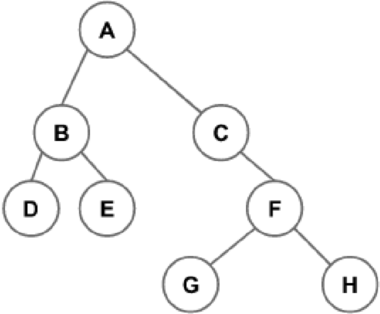
Em *árvores binárias ordenadas* de raiz R, a primeira subárvore de cada nó é denominada *subárvore à esquerda de R* (cuja raiz se chama *filho esquerdo*), e a segunda é a *subárvore à direita de R* (cuja raiz se chama *filho direito*).



Exemplo: B é filho esquerdo e C é filho direito de A

Estritamente *m*-ária

Uma árvore *estritamente m-ária* é aquela na qual cada nó possui *exatamente* 0 ou *m* filhos.

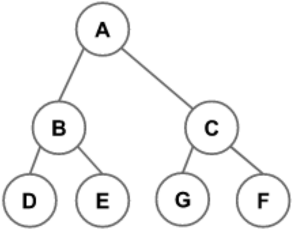


Exemplo: Considere a inclusão de um filho à esquerda de C.

Observação: Chamada pelo NIST de full binary tree, embora também seja preferivelmente chamada de *própria* (ou *proper*).

Árvore Cheia ou Perfeita

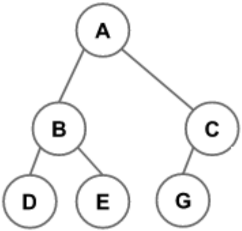
Uma *árvore m-ária cheia* (ou *perfeita*) é aquela na qual todo nó com alguma subárvore vazia está no *último nível*.



Observação: Chamada pelo NIST de perfect binary tree (ou *perfect k-ary tree*), embora também seja chamada de *full* ou, preferencialmente *perfect*.

Árvore Completa

Uma *árvore m-ária completa* é aquela na qual todo nó com alguma subárvore vazia está no *último ou penúltimo níveis, estando os nós do último nível completamente preenchidos da esquerda para a direita*.



Observação: Chamada pelo NIST de complete binary tree. Note que alguns autores consideram essa mesma definição para árvores cheias ou perfeitas. *O ponto fundamental é a facilidade de implementação em vetores (vide próximos slides).* [Knuth97][1]

[1][Knuth97] Donald E. Knuth, The Art of Computer Programming, Addison-Wesley,

Desafios

1. Qual a altura máxima de uma árvore binária com n nós?
2. Qual a altura máxima de uma árvore estritamente binária com n nós?
3. Qual a altura mínima de uma árvore binária com n nós?
4. Numa árvore binária cheia com n nós, qual o número de nós no último nível?

Desafios

1. Qual a altura máxima de uma árvore binária com n nós?
2. Qual a altura máxima de uma árvore estritamente binária com n nós?
3. Qual a altura mínima de uma árvore binária com n nós?
4. Numa árvore binária cheia com n nós, qual o número de nós no último nível?

Solução: n, $(n+1)/2$, $\lceil lg(n+1) \rceil$, $(n+1)/2$

Section 3

Implementações

Implementações de Árvores

Apresentaremos dois tipos de implementação para o TAD Árvore: Sequencial e Encadeada.

Note que, nesse momento, não apresentaremos *operações* sobre o TAD Árvore, focando somente em sua *representação interna*. A razão é que existem diversos tipos específicos de árvores, que apresentam operações distintas no TAD, de acordo com seu propósito.

Implementação Encadeada 1 (*m*-ária)

Consideramos uma implementação de árvore *m*-ária, com alocação encadeada de nós (alocação interna sequencial para filhos).

```
constexpr int M = 3;      // aridade M=3 (ternária)
class NoEnc1
{
public:
   char chave             // dado armazenado
   NoEnc1* nosFilhos[M]; // ponteiros para filhos
};

class ArvoreEnc1
{
public:
  NoEnc1* raiz;           // raiz da árvore
};
```

Implementação Encadeada 1 (*m*-ária)

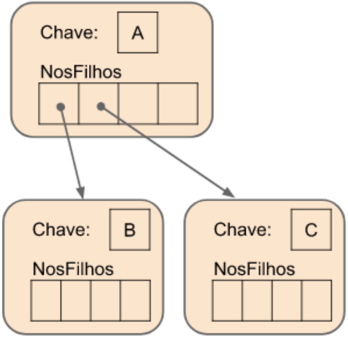


Figure 2: Ilustração `NoEnc1`. Crédito: Fabiano Oliveira

Implementação Encadeada 2 (m-ária)

Consideramos uma implementação de árvore m-ária, com alocação encadeada de nós.

```
constexpr int M = 3;        // aridade M=3 (ternária)
class NoEnc2
{
public:
   char chave;              // dado armazenado
   NoEnc2* prox;            // proximo elemento
   NoEnc2* nosFilhos;       // ponteiro único para filhos
};

class ArvoreEnc2
{
public:
  NoEnc2* raiz;             // raiz da árvore
};
```

Implementação Encadeada 2 (*m*-ária)

Consideramos uma implementação de árvore *m*-ária, com alocação encadeada de nós.

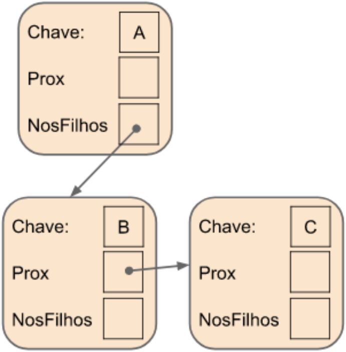


Figure 3: Ilustração `NoEnc2`. Crédito: Fabiano Oliveira

Implementação Encadeada 3 (binária)

Note que podemos reescrever os ponteiros de `NoEnc2` com os termos `esq` e `dir` (nó esquerdo e nó direito).

```
class NoEnc3
{
public:
   char chave;      // dado armazenado
   NoEnc3* esq;     // filho esquerdo
   NoEnc3* dir;     // filho direito
};

class ArvoreEnc3
{
public:
  NoEnc3* raiz;     // raiz da árvore
};
```

Implementação Encadeada 3 (binária)

Consideramos uma implementação de árvore binária, com alocação encadeada de nós.

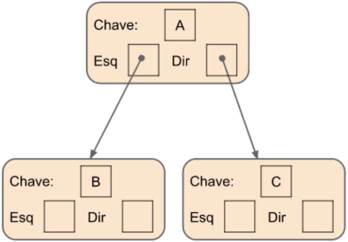


Figure 4: Ilustração `NoEnc3`. Crédito: Fabiano Oliveira

Conversão para Árvores Binárias

Observamos pelas implementações `NoEnc2` e `NoEnc3` que uma árvore *m*-ária *qualquer* pode ser convertida para uma árvore binária. Isso reforça a importância do estudo de implementações eficientes para árvores binárias.

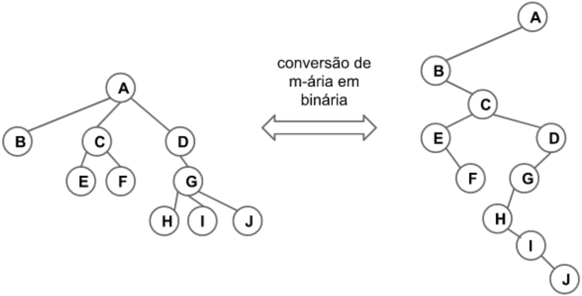


Figure 5: Conversão de Aridade. Crédito: Fabiano Oliveira

Implementação Sequencial

As Árvores com Implementação Sequencial utilizam um array para armazenar os dados. Assim, os dados sempre estarão em um *espaço contíguo* de memória.

Desafio: quanto espaço é necessário para armazenar uma *árvore qualquer* com altura h?

Implementação ArvoreSeq1

Consideraremos uma árvore sequencial com, no máximo, `MAX_N` elementos do tipo caractere.

```
constexpr int MAX_N = 50; // capacidade máxima da árvore
class ArvoreSeq1
{
public:
  char elem [MAX_N];        // elementos na fila
};
```

Desafio: Quantos níveis *cabem* nessa árvore? $\lceil log_2(50+1) \rceil = 6$

Desafios na ArvoreSeq1

Note que, para esse fim, somente as *árvores completas* terão maior eficiência, utilizando uma *representação por níveis*.

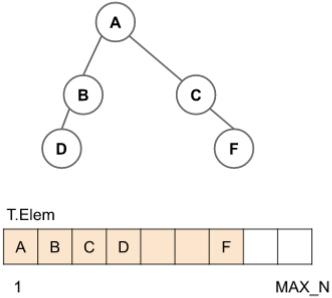


Figure 6: Representação por Níveis. Crédito: Fabiano Oliveira

Desafio: onde fica o primeiro elemento de cada nível da árvore T?

Localização na ArvoreSeq1

Dado um nó V na posição i da árvore sequencial T, em que posição estão:

- o pai de V?
- os filhos de V?

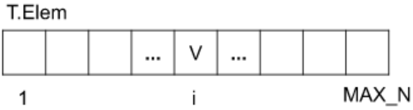


Figure 7: Localização por Níveis. Crédito: Fabiano Oliveira

Localização na ArvoreSeq1

Dado um nó V na posição i da árvore sequencial T, em que posição estão:

- o pai de V?
- os filhos de V?

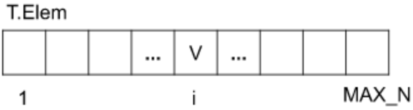


Figure 7: Localização por Níveis. Crédito: Fabiano Oliveira

Resposta: considerando contagem 1..MAX_N, estarão respectivamente nas posições $\lfloor i/2 \rfloor$ (pai), $2i$ e $2i+1$ (filhos).

Desafio: considere a contagem 0..MAX_N-1 e refaça o cálculo.

Localização na ArvoreSeq1

Dado um nó V na posição i da árvore sequencial T, em que posição estão:

- o pai de V?
- os filhos de V?

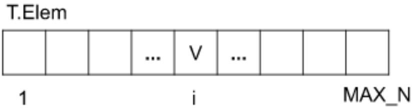


Figure 7: Localização por Níveis. Crédito: Fabiano Oliveira

Resposta: considerando contagem 1..MAX_N, estarão respectivamente nas posições $\lfloor i/2 \rfloor$ (pai), $2i$ e $2i+1$ (filhos).

Desafio: considere a contagem 0..MAX_N-1 e refaça o cálculo.

Solução: posições $\lfloor (i-1)/2 \rfloor$ (pai), $2i+1$ e $2i+2$ (filhos).

Fim implementações

Fim parte de implementações.

Section 4

Percursos em Árvores

Percursos em Árvores

Section 5

Árvores de Busca

Árvores de Busca

Bibliografia Recomendada

Além da bibliografia do curso, recomendamos para esse tópico:

- Szwarcfiter, J.L; Markenzon, L. Estruturas de Dados e seus Algoritmos. Rio de Janeiro, LTC, 1994. Bibliografia Adicional:
- Cerqueira, R.; Celes, W.; Rangel, J.L. Introdução a estruturas de dados: com técnicas de programação em C. Editora, 2004.
- Cormen, T.H.; Leiserson, C.E.; Rivest, R.L.; Stein Algoritmos: Teoria e Prática. Ed. Campus, 2002.
- Cormen, T.H.; Leiserson, C.E.; Rivest, R.L.; Stein, C. Introduction to Algorithms, 3rd ed.. The MIT Press, 2009.
- Preiss, B.R. Estruturas de Dados e Algoritmos Ed. Campus, 2000;
- Knuth, D.E. The Art of Computer Programming - Vols I e III. 2nd Edition. Addison Wesley, 1973.
- Graham, R.L., Knuth, D.E., Patashnik, O. Matemática Concreta. Segunda Edição, Rio de Janeiro, LTC, 1995.
- Livro "The C++ Programming Language" de Bjarne Stroustrup
- Dicas e normas C++: https://github.com/isocpp/CppCoreGuidelines

Section 6

Agradecimentos

Pessoas

Em especial, agradeço aos colegas que elaboraram bons materiais, como o prof. Fabiano Oliveira (IME-UERJ), e o prof. Jayme Szwarcfiter cujos conceitos formam o cerne desses slides.

Estendo os agradecimentos aos demais colegas que colaboraram com a elaboração do material do curso de Pesquisa Operacional, que abriu caminho para verificação prática dessa tecnologia de slides.

Software

Esse material de curso só é possível graças aos inúmeros projetos de código-aberto que são necessários a ele, incluindo:

- pandoc
- LaTeX
- GNU/Linux
- git
- markdown-preview-enhanced (github)
- visual studio code
- atom
- revealjs
- groomit-mpx (screen drawing tool)
- xournal (screen drawing tool)
- ...

Empresas

Agradecimento especial a empresas que suportam projetos livres envolvidos nesse curso:

- github
- gitlab
- microsoft
- google
- ...

Reprodução do material

Esses slides foram escritos utilizando pandoc, segundo o tutorial ilectures:

- https://igormcoelho.github.io/ilectures-pandoc/

Exceto expressamente mencionado (com as devidas ressalvas ao material cedido por colegas), a licença será Creative Commons.

Igor Machado Coelho

This Slide Is Intentionally Blank (for goomit-mpx)