MC202 - Estruturas de Dados

Guilherme P. Telles

IC

8 de maio de 2023

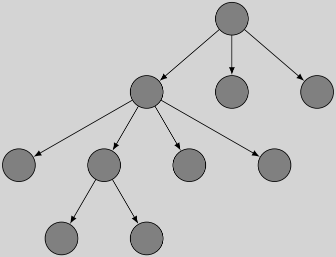
Avisos

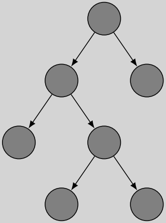
- Estes slides contêm erros.
- Estes slides são incompletos.
- Estes slides usam português anterior à reforma ortográfica de 2009.

Árvores enraizadas

Árvore enraizada

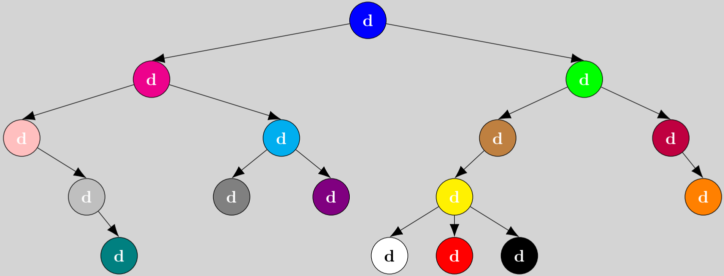
- Um árvore enraizada é uma forma hierárquica de organizar dados na memória:
  - Cada nó armazena dados e apontadores para zero ou mais filhos.





- Na forma mais geral, cada nó de uma árvore pode ter um número qualquer de filhos.
- Os filhos podem ou não ser ranqueados: primeiro filho, segundo filho etc.

Nomenclatura



- Uma **aresta orientada** é a ligação entre **pai** e **filho**.
- A **raiz** da árvore é o nó sem pai.
- Dois nós com o mesmo pai são **irmãos**.

- Um nó que têm pelo menos um filho é um **nó internos**.
- Um nó sem nós como filhos é um **nó externo** ou **folha**.

- Um **caminho** é uma sucessão orientada de zero ou mais arestas

- Várias estruturas de dados têm a forma de uma árvore.
- Árvores binárias com filhos ranqueados são freqüentes.

Árvores binárias

Árvore binária enraizada

- Uma árvore binária enraizada, chamada simplesmente de árvore binária, é formada por nós com dois filhos.
- A ordem dos filhos é importante: um é o filho da esquerda e outro é o filho da direita.

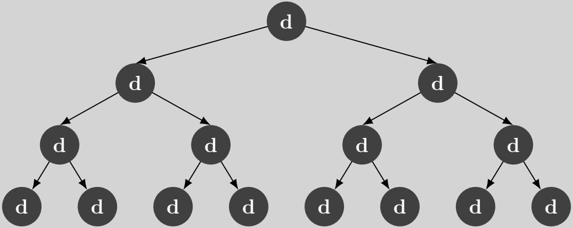
- Uma árvore binária pode ser definida recursivamente da seguinte forma:
  1. Um conjunto vazio de nós é uma árvore binária. A raiz da árvore vazia é nula.
  2. Sejam T_1 e T_2 árvores binárias com raízes r_1 e r_2. Seja r um novo nó. Se r_1 e a r_2 se tornarem filhos de r temos uma árvore binária T com raiz r.

Fatos

- Uma árvore binária não-nula com n nós tem altura máxima $n-1$ e altura mínima $\lfloor \log_2 n \rfloor$.
- Uma árvore binária não-nula com altura h tem no mínimo $h+1$ nós e no máximo $2^{h+1}-1$ nós.

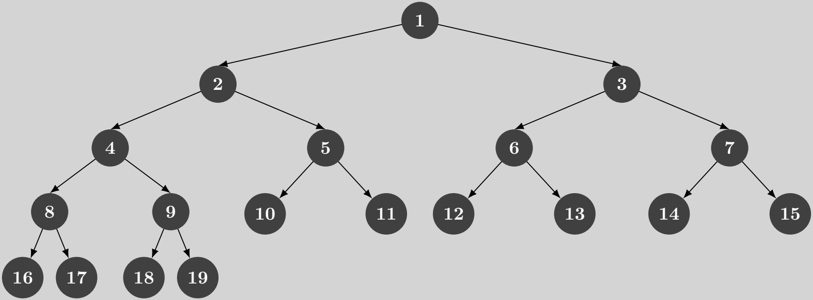
Árvore completa

- Vamos chamar uma árvore binária de **completa** se todos os nós internos têm dois filhos e todas as folhas estão no mesmo nível.



Árvore quase-completa

- Vamos chamar uma árvore binária de **quase-completa** se todos os seus níveis estão preenchidos, exceto talvez pelas folhas à direita do último nível.



- Cada nível ℓ de uma árvore quase-completa, exceto talvez pelo último, tem exatamente 2^ℓ nós.
- Os nós em um nível ℓ podem ser rotulados

$$2^\ell, 2^\ell + 1, 2^\ell + 2, \ldots, 2^{\ell+1} - 1.$$

- O nó i entá no nível $\ell = \lfloor \lg i \rfloor$.
- A altura do nó i é $h = \lfloor \lg \frac{n}{i} \rfloor$.

Percursos em uma árvore binária

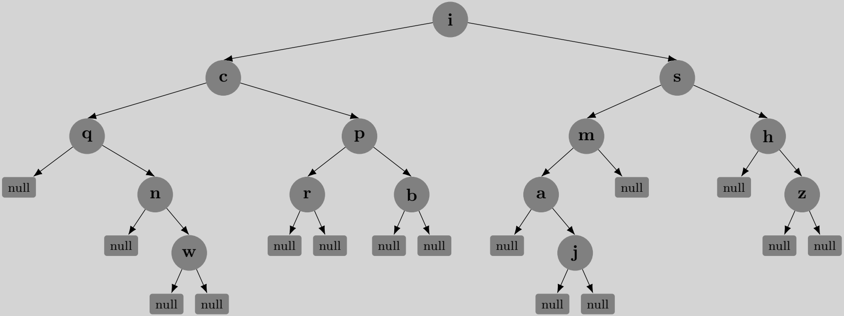
- Um percurso é uma forma de percorrer todos os nós de uma árvore a partir da raiz.
- Alguns percursos são úteis para fornecer informações a respeito da árvore ou dos dados armazenados na árvore.
- Há dois principais:
  1. em profundidade: “para baixo primeiro”.
  2. em largura: “para o lado primeiro”.
- Durante o percurso geralmente realiza-se alguma operação em cada nó, de interesse da aplicação. Vamos chamar essa operação pelo nome genérico **visitar**.

Percursos em profundidade

- São três, definidos recursivamente.
  - pré-ordem:
    1. Visitar a raiz
    2. Percorrer a sub-árvore esquerda
    3. Percorrer a sub-árvore direita
  - em-ordem:
    1. Percorrer a sub-árvore esquerda
    2. Visitar a raiz
    3. Percorrer a sub-árvore direita
  - pós-ordem:
    1. Percorrer a sub-árvore esquerda
    2. Percorrer a sub-árvore direita
    3. Visitar a raiz

Pré-ordem

```
PRE-ORDER(x)
if x ≠ NULL
    visit x
    PRE-ORDER(x.left)
    PRE-ORDER(x.right)
```


i
c
s
q
p
m
h
null
n
r
b
a
null
null
z
null
w
null
null
null
null
null
j
null
null
null
null
null
null

Análise

- O custo da função é constante. Ela é executada uma vez para cada um dos n nós da árvore. Então o tempo é $O(n)$.
- Denotando a altura da árvore por h, no máximo h chamadas de função ficam empilhadas a qualquer momento. Então a memória é $O(h)$.
  - Existem algoritmos que fazem percurs'qos sem usar a pilha, alterando temporariamente os apontadores para indicar "o caminho de volta".'q

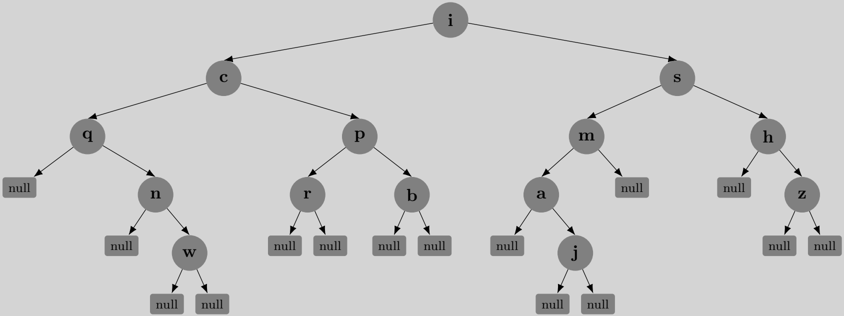
Pré-ordem iterativa

```
PRE-ORDER(x)
Let S be an empty stack
PUSH(S, x)
while S ≠ ∅
    x = POP(S)
    if x ≠ NULL
        visit x
        PUSH(S, x.right)
        PUSH(S, x.left)
```

Em-ordem

$\textsc{In-Order}(x)$

if $x \neq \textsc{Null}$
    $\textsc{In-Order}(x.\mathit{left})$
    visit x
    $\textsc{In-Order}(x.\mathit{right})$


i
c
s
q
p
m
h
null
n
r
b
a
null
null
z
null
w
null
null
null
null
null
j
null
null
null
null
null
null

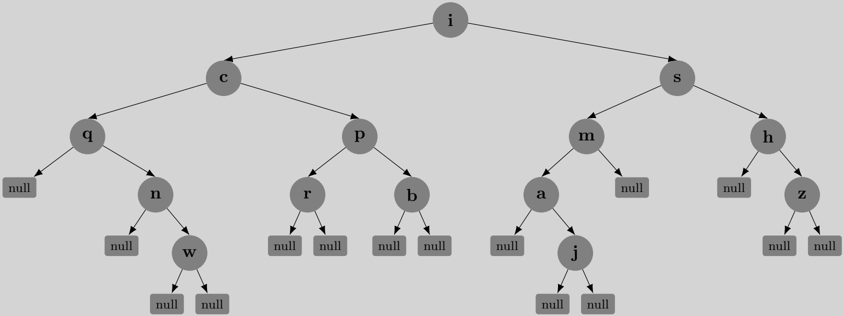
Em-ordem iterativa

```
IN-ORDER(x)
Let S be an empty stack
while IS-NOT-EMPTY(S) or x ≠ NULL
    if x ≠ NULL
        PUSH(S, x)
        x = x.left
    else
        x = POP(S)
        visit x
        x = x.right
```

Pós-ordem

POST-ORDER(x)

if $x \neq$ NULL
    POST-ORDER$(x.\mathit{left})$
    POST-ORDER$(x.\mathit{right})$
    visit x


i
c
s
q
p
m
h
null
n
r
b
a
null
null
z
null
w
null
null
null
null
null
j
null
null
null
null
null
null

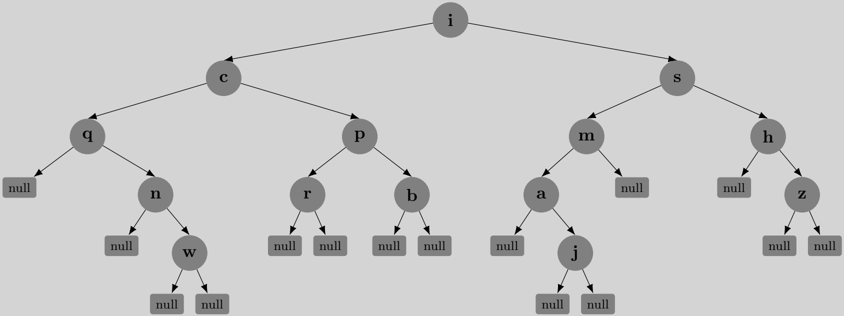
Pós-ordem iterativa

```
POST-ORDER(x)
Let S be an empty stack
while IS-NOT-EMPTY(S)
    while x ≠ NULL
        if x.right ≠ NULL
            PUSH(S, x.right)
        PUSH(S, x)
        x = x.left
    x = POP(S)
    if x.right ≠ NULL and x.right == AT_TOP(S)
        y = POP(S)
        PUSH(S, x)
        x = x.right
    else
        visit x
        x = NULL
```

Percurso em largura

- Visita os nós por níveis.

```
BREADTH(root)
Let Q be an empty queue
ENQUEUE(root)
while Q ≠ ∅
    node p = DEQUEUE(Q)
    visit p
    if p.left ≠ NULL
        ENQUEUE(Q, p.left)
    if p.right ≠ NULL
        ENQUEUE(Q, p.right)
```


i
c
s
q
p
m
h
null
n
r
b
a
null
null
z
null
w
null
null
null
null
null
j
null
null
null
null
null
null

Representação de árvores binárias

Representação explícita ou encadeada

- Cada nó tem os dados e dois apontadores: um para o filho da esquerda e um para o filho da direita.

Representação com apontador para o pai

- Usa mais memória em cada nó, os dados e três apontadores: um para o pai e dois para os filhos.
- Permite percorrer a árvore das folhas para a raiz.
- Os algoritmos para percursos em profundidade não precisam usar uma pilha.

Representação costurada (threaded)

- As folhas têm dois apontadores nulos. São mais numerosos que os demais. (É muita memória apontando para nada.)
- Os apontadores nulos nas folhas podem ser redefinidos assim:
  - o filho da esquerda aponta para o predecessor em-ordem,
  - o filho da direita aponta para o sucessor em-ordem e
  - os extremos na ordem apontam para a raiz.
- O nó precisa registrar se cada apontador é para um filho ou é threaded. Um bit por apontador é suficiente (embora não seja fácil usar apenas 1 bit).

- Dessa forma é possível percorrer os nós em-ordem e em-ordem-inversa sem apontador para o pai e sem usar uma pilha.
- O custo adicional para manter os threads é baixo.
- Pode ser costurada apenas nos apontadores direitos ou apenas nos esquerdos, se apenas uma das ordens for necessária.

Representação implícita ou seqüencial

- Os nós de uma árvore podem ser colocados em um vetor de tal forma que
  1. a raiz está na posição 1 e
  2. os filhos do nó i são $2i$ e $2i + 1$.
- Dessa forma, o pai do nó i está em $\lfloor \frac{i}{2} \rfloor$.
- Se a árvore é quase-completa então essa representação usa pouca memória.

Representação como vetor de predecessores

- Os nós de uma árvore podem ser colocados em um vetor P de tal forma que
  1. $P[i]$ é o índice do pai do nó i.
  2. $P[raiz]$ é o índice da própria raiz.
- Para percorrer um caminho da raiz até um nó i percorremos um caminho de i até a raiz e empilhamos os nós ao longo do caminho.
- Essa representação usa pouca memória mesmo se a árvore não é quase-completa, mas encontrar os nós e percorrê-la leva mais tempo.

Árvores gerais

Árvore enraizada

- Em uma árvore enraizada (ou árvore geral ou simplesmente árvore), cada nó tem k filhos.
- Todos os conceitos de hierarquia e ancestralidade continuam bem definidos em árvores gerais.

Percursos em árvores gerais

- Pré-ordem e pós-ordem continuam bem definidas em árvores gerais.
  - pré-ordem:
    1. Visitar a raiz
    2. Percorrer as subárvores da raiz.
  - pós-ordem:
    1. Percorrer as subárvores da raiz.
    2. Visitar a raiz

Representação explícita

- Cada nó tem um vetor de apontadores para os k filhos dele.
- Se os filhos forem ranqueados então o vetor tem que ter tamanho igual a k ou é necessário registrar o rank de cada filho.

Representação filho-irmão

- Nessa representação cada nó aponta para seu filho mais à esquerda e para seu irmão imediato.
- Se os filhos forem rankeados então cada nó tem que ter um campo indicando qual filho ele é na ordem.

Representação implícita

- Uma árvore com k filhos em cada nó também pode ser representada implicitamente:
  - a raiz está na posição 1 e
  - os filhos do nó i estão nas posições $(i-1)k+2, (i-1)k+3, \ldots, (i-1)k+k+1$.

  Logo o pai do nó i está em $\lfloor \frac{i+1}{k} \rfloor$.

Florestas

Florestas

- Uma floresta é um conjunto de árvores não-vazias.
- Uma forma de representar uma floresta é como uma lista de raízes das árvores.
- A representação filho-irmão pode ser usada para representar florestas: a raiz de cada árvore aponta para o filho da esquerda e para a próxima árvore.

Percursos em florestas

- Pré-ordem e pós-ordem continuam bem definidas em florestas.
  - pré-ordem:
    1. Visitar a raiz da primeira árvore
    2. Percorrer as subárvores da primeira árvore.
    3. Percorrer as demais árvores.
  - pós-ordem:
    1. Percorrer as subárvores da primeira árvore.
    2. Visitar a raiz da primeira árvore
    3. Percorrer as demais árvores.