Árvores

Linguagem Haskell

Maria Adriana Vidigal de Lima

Faculdade de Computação - UFU

Dezembro - 2009

1. Árvores em Haskell
   - Noções sobre Árvores
   - Árvore Binária de Busca
   - Árvore Binária Genérica

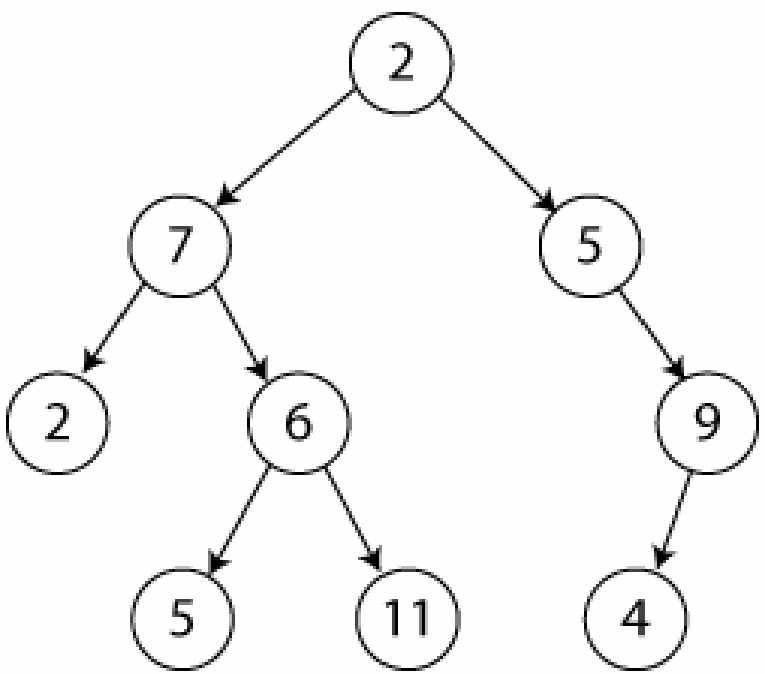
Fundamentos

- Uma árvore é uma estrutura de dados baseada em listas encadeadas, que possuem um elemento superior, definido como a **raiz** da árvore.
- O nó raiz aponta para outros nós, denominados nós filhos. Cada nó filho pode ser nó pai de outros nós abaixo deste. Dessa forma, a árvore é uma estrutura recursiva.
- As **árvores binárias** são um tipo especial de árvore e têm o fator de ramificação limitado em dois. Assim, qualquer nó da árvore binária tem no máximo dois nós filhos.

Exemplo de Árvore Binária

Uma árvore binária é uma estrutura de dados caracterizada por:

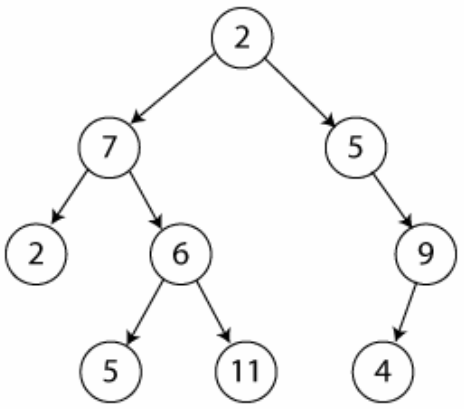
- Não possui elementos (árvore vazia).
- Possui um elemento distinto (raiz), com dois ponteiros para as sub-árvore esquerda e sub-árvore direita.



Árvore Binária em Haskell

A definição de dados para uma árvore binária em Haskell considerando o armazenamento de números inteiros pode ser:

```
data ArvoreBinInt = Nulo |
                    No Int ArvoreBinInt ArvoreBinInt
                       deriving Show
```



A árvore binária da figura é descrita por:

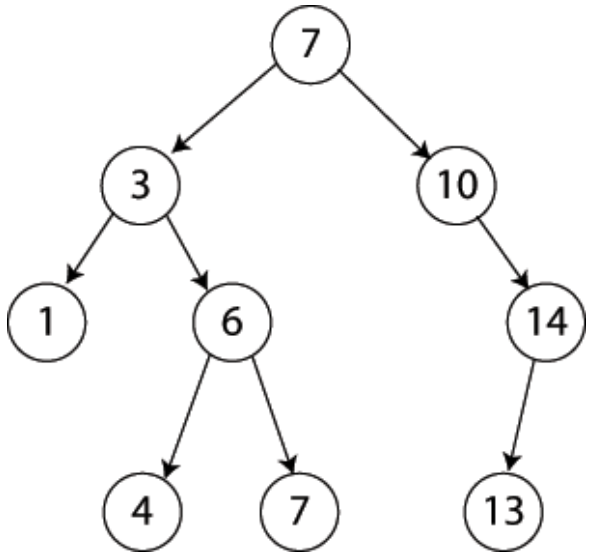
```
arvEx = (No 2 (No 7 (No 2 Nulo Nulo)
                    (No 6 (No 5 Nulo Nulo)
                          (No 11 Nulo Nulo)))
              (No 5  Nulo
                    (No 9 (No 4 Nulo Nulo)
                          Nulo)))
```

Árvore Binária de Busca

Uma árvore binária é de **busca** quando é vazia ou quando satisfaz as propriedades

- o elemento no nó raiz é *maior* que todos os elementos da sub-árvore esquerda.
- o elemento no nó raiz é *menor* que todos os elementos da sub-árvore direita.
- as sub-árvores direita e esquerda também são árvores binárias de busca.

Exemplo de Árvore Binária de Busca



Representação em Haskell:

```
arvEx = (No 7 (No 3 (No 1 Nulo Nulo)
                    (No 6 (No 4 Nulo Nulo)
                          (No 6 Nulo Nulo)))
              (No 10  Nulo (No 14 (No 13 Nulo Nulo) Nulo)))
```

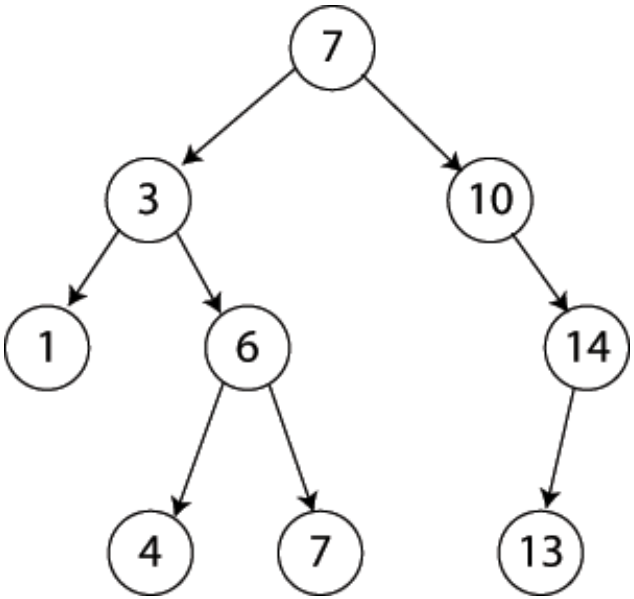
Travessia de uma Árvore Binária de Busca

Um percurso (em ordem) numa árvore binária de busca retorna os elementos numa lista, de forma **ordenada**.

```
data ArvoreBinInt = Nulo |
                    No Int ArvoreBinInt ArvoreBinInt
                       deriving Show

emOrdem :: ArvoreBinInt -> [Int]
emOrdem Nulo = []
emOrdem (No x esq dir) = (emOrdem esq) ++ [x] ++ (emOrdem dir)
```

Travessia emOrdem numa Árvore Binária de Busca




```
arvEx = (No 7 (No 3 (No 1 Nulo Nulo)
                    (No 6 (No 4 Nulo Nulo)
                          (No 6 Nulo Nulo)))
              (No 10  Nulo
                      (No 14 (No 13 Nulo Nulo)
                             Nulo)))
```

Execução da função emOrdem:

```
> emOrdem arvEx
[1,3,4,6,6,7,10,13,14]
```

Definição Genérica para Árvore Binária

```
data ArvBinGen a = Nulo | No a (ArvBinGen a)
                                (ArvBinGen a)
```

Nova exibição da Árvore Binária

```
instance Show a => Show (ArvBinGen a) where
    show Nulo = "_"
    show (No x esq dir) = "{" ++ show x ++ ":" ++
                                  show esq ++ "|" ++
                                  show dir ++ "}"
```

Exemplo:

```
{4: {2: {1: _|_} | {3: _|_} } |
    {6: {5: _|_} | {7: _|_} }
}
```

Inserção de um Elemento numa Árvore Binária de Busca

```
insereGen::Ord a=> a->ArvBinGen a->ArvBinGen a
insereGen x Nulo = (No x Nulo Nulo)
insereGen x (No y esq dir)
    | x == y    = No y esq dir
    | x < y     = No y (insereGen x esq) dir
    | otherwise = No y esq (insereGen x dir)
```

Nesta função insere, os elementos repetidos não serão considerados.

Inserção de um Elemento numa Árvore Binária de Busca

Exemplos

```
> insereGen 5 (No 2 (No (-1) Nulo Nulo) Nulo)
{2:{-1:_|_}|{5:_|_}}

> insereGen 5 (No 2 (No (-1) Nulo Nulo) (No 8 Nulo Nulo))
{2:{-1:_|_}|{8:{5:_|_}|_}}

> insereGen 8 (No 2 (No (-1) Nulo Nulo) (No 8 Nulo Nulo))
{2:{-1:_|_}|{8:_|_}}
```

Remoção de um Elemento numa Árvore Binária de Busca

```
data ArvBinGen a = Nulo | No a (ArvBinGen a)
                                (ArvBinGen a) deriving Eq
                                ----------
```

```
removeGen::Ord a=> a->ArvBinGen a->ArvBinGen a
removeGen val Nulo = Nulo
removeGen val (No v esq dir)
     | val < v = No v (removeGen val esq) dir
     | val > v = No v esq (removeGen val dir)
     | esq == Nulo = dir
     | dir == Nulo = esq
     | otherwise = juntaGen esq dir
```

Remoção de um Elemento numa Árvore Binária de Busca

```
juntaGen::Ord a=> ArvBinGen a->ArvBinGen a->ArvBinGen a
juntaGen esq dir = No menor esq novadir
     where
          menor = menorGen dir
          novadir = removeGen menor dir

menorGen::Ord a=>ArvBinGen a->a
menorGen Nulo = undefined
menorGen (No x esq dir)
    | esq == Nulo = x
    | otherwise = menorGen esq
```

Balanceamento em uma Árvore Binária de Busca

```
data ArvBinGen a = Nulo | No a (ArvBinGen a)
                                (ArvBinGen a) deriving Eq
```

Uma Árvore de Busca Binária tem seus elementos ordenados e é:

- eficiente quando a árvore está balanceada
- ineficiente se a árvore se assemelha à uma lista

Balanceamento: Árvore AVL

- Uma árvore é dita **balanceada** quando as suas subárvores à esquerda e à direita possuem a mesma altura, ou no máximo com diferença de um nível.
- Se todos os nós folha estiverem no mesmo nível, então a árvore é completa.

Bibliografia

1. *Haskell - Uma abordagem prática*. Cláudio César de Sá e Márcio Ferreira da Silva. Novatec, 2006.
2. *Haskell - The craft of functional programming*. Simon Thompson. Pearson, 1999.