INSTITUTO FEDERAL DE ALAGOAS – IFAL / CAMPUS MACEIÓ
Curso Bacharelado em Sistemas de Informação
Disciplina de Estruturas de Dados

Prof. MSc. Ricardo Nunes
Ricardo (arroba) ifal.edu.br

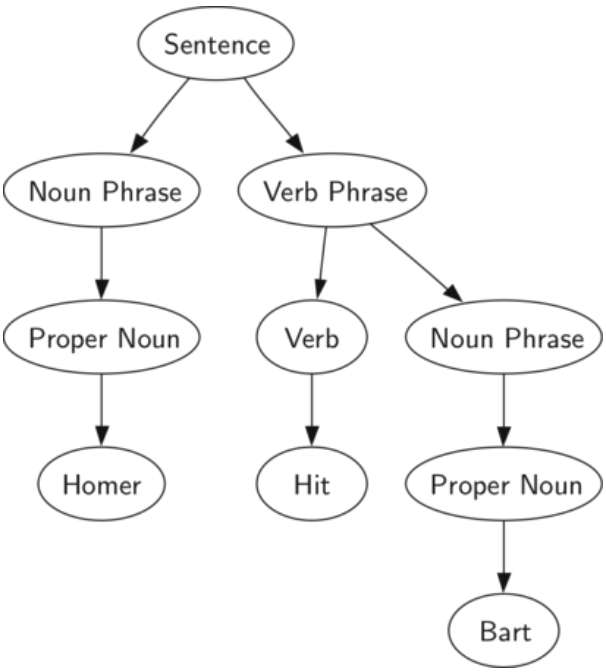
Árvores (2)

Objetivos

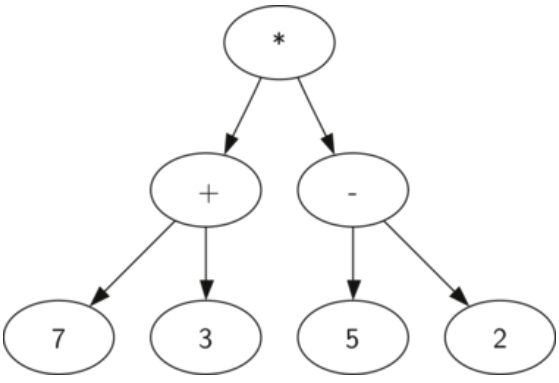
- Implementar algoritmos que manipulam árvores
- Implementar árvores como uma estrutura de dados recursiva

Árvores de Análise (Parse Tree)

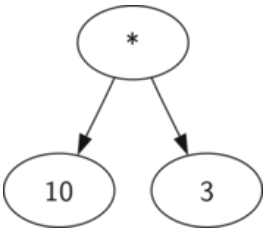
Como utilizar as implementações estudadas em problemas reais?



Exemplo



((7+3)□(5−2))



Interesses

- Como construir uma árvore de análise a partir de uma expressão matemática parentizada
- Como avaliar a expressão armazenada na árvore de análise
- Como recuperar a expressão matemática original a partir de uma árvore de análise

Como construir uma árvore de análise a partir de uma expressão matemática parentizada

- Passo 1: quebrar a expressão em uma lista de tokens
- Tipos de tokens e ações a serem executadas:
  - parênteses esquerdos: inicia uma nova expressão criando uma nova árvore correspondente
  - parênteses direito: termina uma expressão
  - operadores: terão os filhos esquerdo e direito
  - operandos: serão folhas e filhos dos operadores

Regras

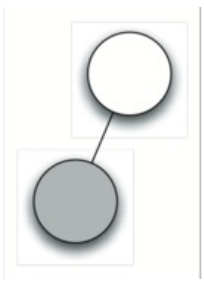
1. Se o token é um '(', adicione um novo nó como o filho esquerdo do nó corrente, e desça até o filho esquerdo
2. Se o token está na lista ['+','-','/','*'], altere o valor raiz do nó corrente para o operador represenado pelo token. Adicione um novo nó como o filho direito do nó corrente e desça até o filho direito.
3. Se o token é um número, altere o valor raiz do nó corrente para o número e retorne ao pai
4. Se o token corrente é um ')', vá par ao pai do nó corrente.

Processando uma expressão

- Expressão: (3+(4□5))
- Lista de tokens: ['(', '3', '+', '(', '4', '*', '5' ,')',')']

Processando uma expressão

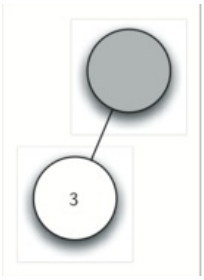
- Expressão: (3+(4□5))
- Lista de tokens: ['(', '3', '+', '(', '4', '*', '5' ,')',')']
- token atual: (



Processando uma expressão

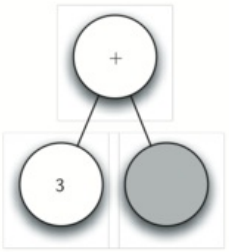
- Expressão: (3+(4□5))

- Lista de tokens: ['(', '3', '+', '(', '4', '*', '5' ,')',')']
- token atual: 3



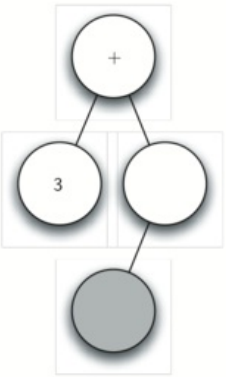
Processando uma expressão

- Expressão: (3+(4□5))
- Lista de tokens: ['(', '3', '+', '(', '4', '*', '5' ,')',')']
- token atual: +



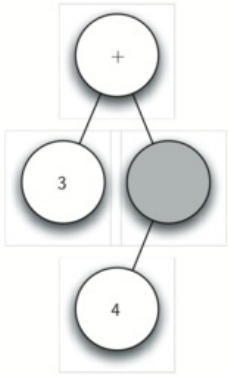
Processando uma expressão

- Expressão: (3+(4□5))
- Lista de tokens: ['(', '3', '+', '(', '4', '*', '5' ,')',')']
- token atual: (



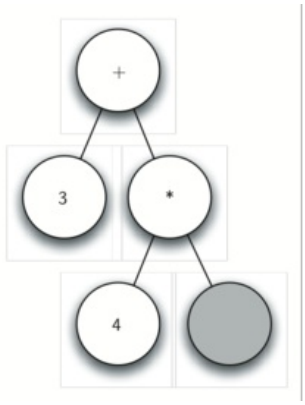
Processando uma expressão

- Expressão: (3+(4□5))
- Lista de tokens: ['(', '3', '+', '(', '4', '*', '5' ,')',')']
- token atual: 4



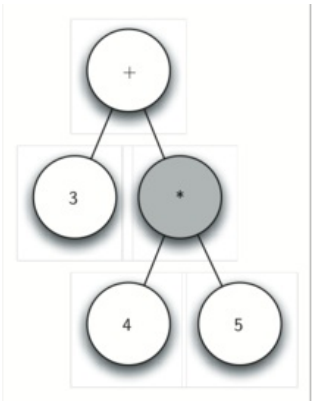
Processando uma expressão

- Expressão: (3+(4□5))
- Lista de tokens: ['(', '3', '+', '(', '4', '*', '5' ,')',')']
- token atual: *



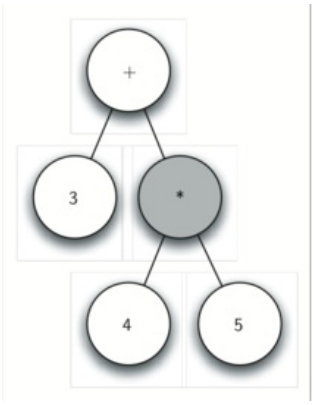
Processando uma expressão

- Expressão: (3+(4□5))
- Lista de tokens: ['(', '3', '+', '(', '4', '*', '5' ,')',')']
- token atual: 5



Processando uma expressão

- Expressão: (3+(4□5))
- Lista de tokens: ['(', '3', '+', '(', '4', '*', '5' ,')',')']
- token atual:)



Considerações:

- É preciso registrar o nó atual e o seu pai
- A interface tem as funções **getLeftChild** e **getRightChild**, mas como recuperar os pais?

-- + Usando uma pilha para guardar os pais conforme a árvore é percorrida

```
#parseTreeEx1.py
from pilha_array import PilhaArray
from binaryTreeEx2 import BinaryTree
def buildParseTree(fpexp):
    fplist = fpexp.split()
    pStack = PilhaArray()
    eTree = BinaryTree('')
    pStack.push(eTree)
    currentTree = eTree
    for i in fplist:
        if i == '(':
            currentTree.insertLeft('')
            pStack.push(currentTree)
            currentTree = currentTree.getLeftChild()
        elif i not in ['+', '-', '*', '/', ')']:
            currentTree.setRootVal(int(i))
            parent = pStack.pop()
            currentTree = parent
        elif i in ['+', '-', '*', '/']:
            currentTree.setRootVal(i)
            currentTree.insertRight('')
            pStack.push(currentTree)
            currentTree = currentTree.getRightChild()
        elif i == ')':
            currentTree = pStack.pop()
        else:
            raise ValueError
    return eTree
if __name__ == "__main__":
  pt = buildParseTree("( ( 10 + 5 ) * 3 )");  #print(pt)
  pt.postorder()
```

.center[E agora?]

- Avaliar a expressão, retornando o resultado
- Pode-se utilizar um algoritmo recursivo que avalia uma árvore analisando cada subárvore recursividamente

Avaliando uma expressão

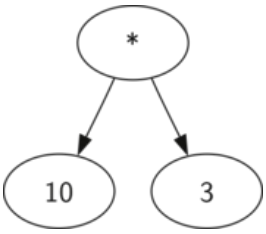
```
def evaluate(parseTree):
    import operator

    opers = {'+':operator.add, '-':operator.sub,
             '*':operator.mul, '/':operator.truediv }

    leftC = parseTree.getLeftChild()
    rightC = parseTree.getRightChild()

    if leftC and rightC:
        fn = opers[parseTree.getRootVal()]
        return fn(evaluate(leftC),evaluate(rightC))
    else:
        return parseTree.getRootVal()
```

Exemplo

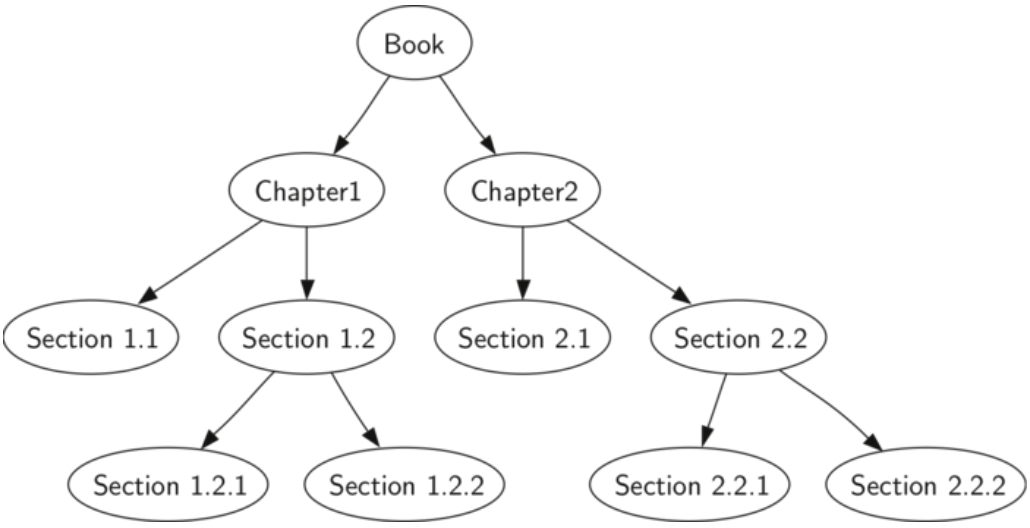


Percorrendo uma árvore

Há três formas de acessar os nós de uma árvore

1. pré-ordem: visita a raiz e depois, recursivamente, percorre as subárvores com acesso pré-ordem, primeiro esquerda, depois direita
2. em-ordem: percorre "em-ordem" recursivamente o nó filho da esquerda, acessa a raiz, e depois percorre "em-ordem" recursivamente o nó filho da direita
3. pós-ordem: percorre "pós-ordem" recursivamente o nó filho da esquerda, depois o nó filho da direita e depois acesa a raiz.

Percorrendo uma árvore - Exemplo



- Pré-ordem: Book > Chapter 1 > Section 1.1 > Section 1.2 > Section 1.2.1 > ...
- Em-ordem: Section 1.1 > Chapter 1 > Chapter 1.2.1 > Section 1.2 > Section 1.2.2 > ...
- Pré-ordem: Section 1.1 > Section 1.2.1 > Section 1.2.2 > Section 1.2 > Chapter 1 > ...

Pré-ordem

Função

```
def preorder(tree):
    if tree:
        print(tree.getRootVal())
        preorder(tree.getLeftChild())
        preorder(tree.getRightChild())
```

Método

```
def preorder(self):
    print(self.key)
    if self.leftChild:
        self.leftChild.preorder()
    if self.rightChild:
        self.rightChild.preorder()
```

Pós-ordem

Função

```
def postorder(tree):
    if tree != None:
        postorder(tree.getLeftChild())
        postorder(tree.getRightChild())
        print(tree.getRootVal())
```

Avaliando uma expressão

```
def postordereval(tree):
    import operator
    opers = {'+':operator.add, '-':operator.sub, '*':operator.mul, '/':operator.truediv}
    res1 = None
    res2 = None
    if tree:
        res1 = postordereval(tree.getLeftChild())
        res2 = postordereval(tree.getRightChild())
        if res1 and res2:
            return opers[tree.getRootVal()](res1,res2)
        else:
            return tree.getRootVal()
```

Em-ordem

Função

```
def inorder(tree):
  if tree != None:
      inorder(tree.getLeftChild())
      print(tree.getRootVal())
      inorder(tree.getRightChild())
```

```
def printexp(tree):
  sVal = ""
  if tree:
      sVal = '(' + printexp(tree.getLeftChild())
      sVal = sVal + str(tree.getRootVal())
      sVal = sVal + printexp(tree.getRightChild())+')'
  return sVal
```

Para estudar

- Árvores
  - Seções 7.6 e 7.7 do livro [6] https://runestone.academy/ns/books/published//pythonds/Trees/toctree.html (em inglês)
  - Capítulo sobre árvores em qualquer livro sobre estruturas de dados

Referências

1. Tradução do livro *How to Think Like a Computer Scientist: Interactive Version*, de Brad Miller e David Ranum. link: https://panda.ime.usp.br/pensepy/static/pensepy/index.html
2. Allen Downey, Jeff Elkner and Chris Meyers. *Aprenda Computação com Python 3.0*. link: https://chevitarese.files.wordpress.com/2009/09/aprendacomputaocompython3k.pdf
3. SANTOS, A. C. *Algoritmo e Estrutura de Dados I*. 2014. Disponível em http://educapes.capes.gov.br/handle/capes/176522
4. SANTOS, A. C. *Algoritmo e Estrutura de Dados II*. 2014. Disponível em 2014. Disponível em https://educapes.capes.gov.br/handle/capes/176557
5. Tradução do livro [6] *Problem Solving with Algorithms and Data Structures using Python* de Brad Miller and David Ranum. link: https://panda.ime.usp.br/pythonds/static/pythonds_pt/index.html
6. Brad Miller and David Ranum. *Problem Solving with Algorithms and Data Structures using Python* link: https://runestone.academy/ns/books/published//pythonds/index.html
7. Caelum. Algoritmos e Estruturas Dados em Java. Disponível em https://www.caelum.com.br/download/caelum-algoritmos-estruturas-dados-java-cs14.pdf

That's all Folks