Trees
Data Structures and Algorithms for Computational Linguistics III (ISCL-BA-07)

Çağrı Çöltekin

ccoltekin@sfs.uni-tuebingen.de

University of Tübingen
Seminar für Sprachwissenschaft

Winter Semester 2021/22

version 695464 6397:17:92

Why study trees

- A tree is a *hierarchical, non-linear* data structure useful in many algorithms
- We have already resorted to descriptions using trees (e.g., recursion trace)
- A tree is a *graph* with certain properties
- It is also very common in (computational) linguistics:
    - Parse trees: representing syntactic structure of sentences
    - Language trees: representing the historical relations between languages
    - Decision trees: a well-known algorithm for machine learning, also used for many NLP problems


Definitions

- A tree is a set of **nodes** organized hierarchically with the following properties:
    - If a tree is non-empty, it has a special node the **root**
    - Except the root node, every node in the tree has a unique **parent** (all nodes except the root are **children** of another node)
- Alternatively, we can define a tree recursively:
    - The empty set of nodes is a tree
    - Otherwise a tree contains a root with sub-trees as its children

---
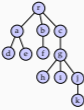
More definitions

- The nodes with the same parent are called siblings
- The nodes with children are called internal nodes
- The nodes without children are the leaf nodes
- A path is a sequence of connected nodes
- Any node in the path from the root to a particular node is its ancestors
- A node is the descendant of its ancestors
- The subtree is a tree rooted by a non-root node
- The depth of a node is the number of edges from root
- The height of a node is the number of edges from the deepest descendant
- The height of a tree is the height of its root
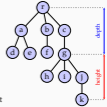
Ordered trees

- A tree is ordered if there is an ordering between siblings. Typical examples include:
    - A tree representing a document (e.g., HTML) structure
    - Parse trees
    - (maybe) a family tree
- In many cases order is not important
    - Class hierarchy in an object-oriented program
    - The tree representing files in a computer

Binary trees
even more definitions

- Binary trees, where nodes have at most two children, have many applications
- Binary trees have a natural order, each child is either a *left child* or a *right child*
- A tree is *proper*, or *full* if every node has either two children or none
- In a *complete* binary tree, every level except possibly the last, is completely filled, and all nodes at the last level is at the left
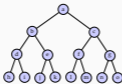- A *perfect* binary tree is a full binary tree whose leaf nodes have the same depth

---

Some properties of binary trees

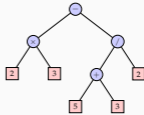For a binary tree with n_L leaf, n_I internal, n nodes and with height h

- $h + 1 \leqslant n \leqslant 2^{h+1} - 1$
- $1 \leqslant n_L \leqslant 2^h$
- $h \leqslant n_I \leqslant 2^h - 1$
- $\log(n + 1) - 1 \leqslant h \leqslant n - 1$
- For any proper binary tree, $n_I = n_L + 1$
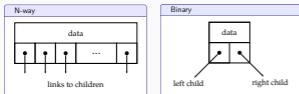

Binary tree example: expression trees
$2 \times 3 + (5 + 3)/2$

Implementation of trees
general case: linked data structures

N-way

data

links to children

Binary

data

left child   right child

Implementation of trees
array implementation of binary trees

- Binary trees can also be implemented with arrays:
    - the root node is stored at index 0
    - the left child of the node at index i is stored at $2i + 1$
    - the right child of the node at index i is stored at $2i + 2$
    - the parent of the node at index i is at index $\lfloor (i-1)/2 \rfloor$
- If the binary tree is complete, this representation does not waste (much) space

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

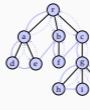## Breadth first traversal (level order)

```python
def breadth_first(root):
    queue = []
    queue.append(root)
    while queue:
        node = queue.pop(0)
        # process the node
        print(node.data)
        for child in node.children:
            queue.append(child)
```
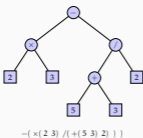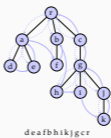
r a b c d e f g h i j k

Pre-order traversal

```python
def pre_order(node):
    # process the node
    print(node.data)
    for child in node.children:
        pre_order(child)
```

r a d e b f c g h i j k

Example: pre-order in an expression tree

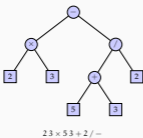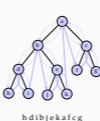$$-(\times(2\ 3)\ /(+(5\ 3)\ 2)\ )\ )$$

Post-order traversal


```python
def post_order(node):
    for child in node.children:
        post_order(child)
    # process the node
    print(node.data)
```

d e a f b h i k j g c r

Example: post-order in an expression tree

$$2\ 3\ \times\ 5\ 3\ +\ 2\ /\ -$$

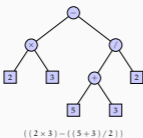## In-order traversal


```python
def in_order(node):
    in_order(node.left)
    # process the node
    print(node.data)
    in_order(node.right)
```

h d i b j e k a f c g

Example: in-order in an expression tree

$$(\ (\ 2 \times 3\ ) - (\ (\ 5 + 3\ ) / 2\ )\ )$$

Summary

- Trees are hierarchical data structures useful in many applications
- We will often return to trees and properties of trees in the rest of the course
- Reading on trees: Goodrich, Tamassia, and Goldwasser (2013, chapter 8), and optionally the chapter on *search trees* (Goodrich, Tamassia, and Goldwasser 2013, ch. 11)

Next:
- Heaps and priority queues
- Reading: Reading: Goodrich, Tamassia, and Goldwasser (2013, chapter 9)

Acknowledgments, credits, references

Goodrich, Michael T., Roberto Tamassia, and Michael H. Goldwasser (2013). *Data Structures and Algorithms in Python*. John Wiley & Sons, Incorporated. ISBN: 9781118476734.