Organising information: trees

Author(s)
Silvio Peroni – silvio.peroni@unibo.it – https://orcid.org/0000-0003-0530-4305
Digital Humanities Advanced Research Centre (DHARC), Department of Classical Philology and Italian Studies, University of Bologna, Bologna, Italy

Keywords
Buendia family; Gabriel García Márquez; Markup language; Tree

Abstract

This chapter introduces a new data structure for defining hierarchical relations: the tree. The historic hero introduced in these notes is Gabriel García Márquez, one of the most notable writers in Spanish of the 20th century. One of his novels (*One Hundred Years of Solitude*) is used in this chapter to introduce the way trees (as a data structure) can be used to understand a story. Moreover, even to structure a text.

Historic hero: Gabriel García Márquez

Gabriel García Márquez (shown in Figure 1) was a Colombian novelist, and he was one of the most notable writers in Spanish of the 20th century. He won the Nobel Prize for Literature in 1982. As a journalist, he wrote several non-fictional works. However, he is mainly known for his novels, such as *Cien años de soledad (One Hundred Years of Solitude* in English) and *El amor en los tiempos del cólera* (*Love in the Time of Cholera* in English).

Several of his works mention the fictional town of Macondo. This city is the primary setting of one of his books, i.e. *One Hundred Years of Solitude*, which narrates the story of the Buendia family. In particular, the story introduces the life of seven different generations of people of the same family. It follows its adventures and, often, misfortunes.

In the Italian edition of this novel, published by Mondadori [García Márquez, 1967], the publisher inserted a family tree of the various generations at the beginning of the book. While it provides a few spoilers about future events, it is instrumental in following the family's story since several Buendia people often share the same name. Thus, the family tree is a handy device that facilitates the reader to follow the story, understanding to whom the narrator is referring.

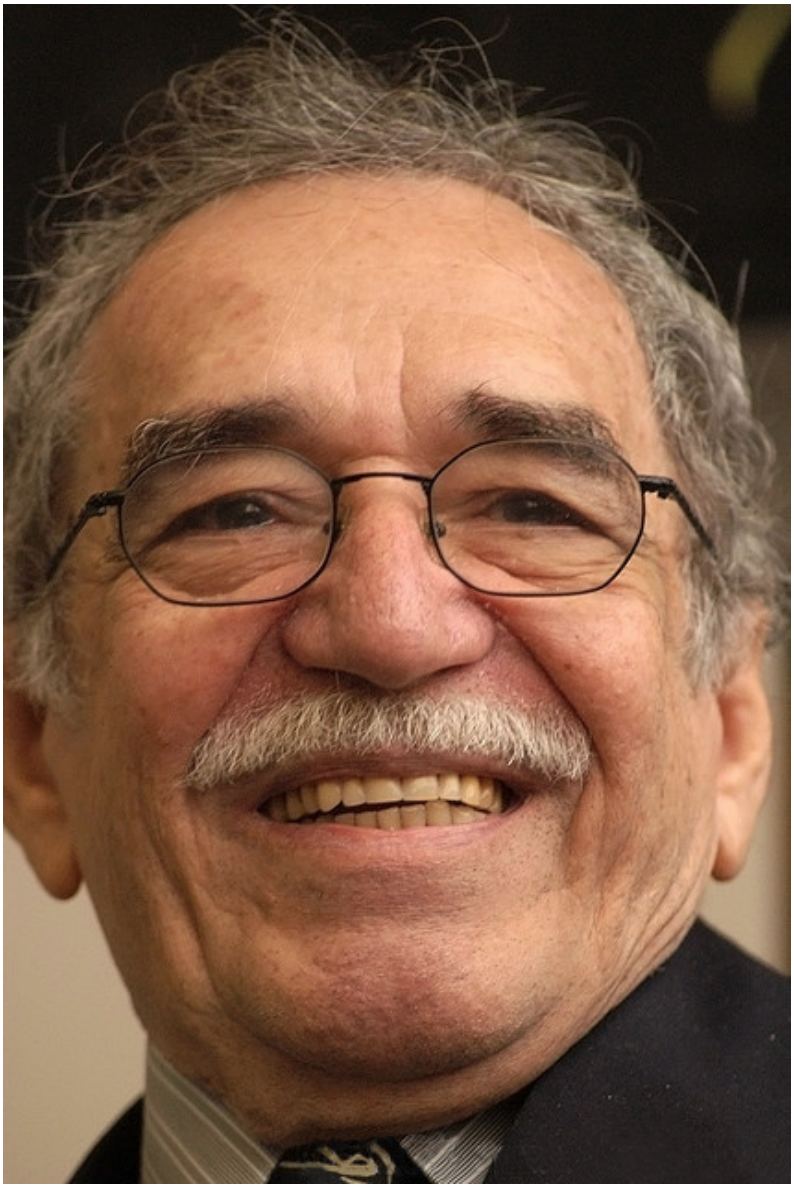

Figure 1. A portrait of Gabriel García Márquez. Picture by Jose Lara, source: https://en.wikipedia.org/wiki/File:Gabriel_Garcia_Marquez.jpg.

García Márquez is the second Latin American writer we have used in this course. We use him to introduce specific topics related to the Computational Thinking and Computer Science domains. For example, trees (such as family trees) are particular kinds of structures that allow one to define a hierarchy of values useful for a plethora of different tasks or computations.

Where is the tree?

It is not the first time we have adopted a tree for describing something. In particular, in the chapter "Dynamic programming algorithms", we used a tree (reprised in Figure 2) to show the recursive calls' execution to the function implementing the algorithm to find the Fibonacci number.

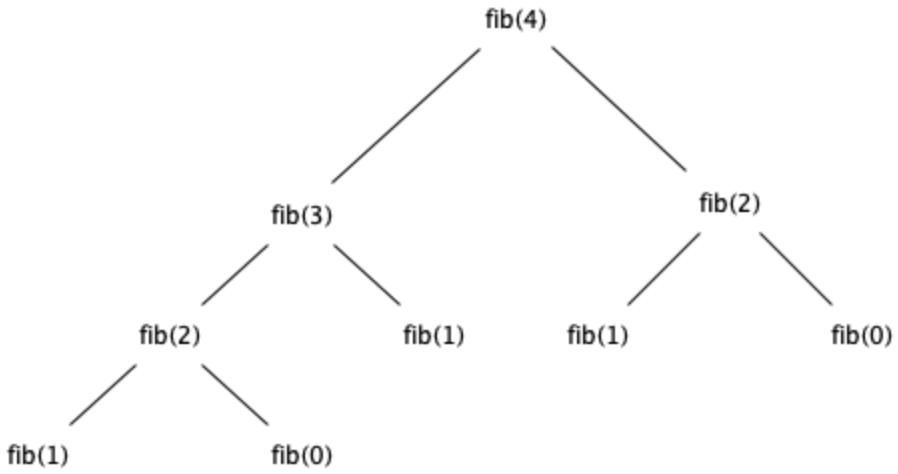

Figure 2. The tree that describes the various recursive calls for calculating the Fibonacci number at the 4th month, as described in chapter "Dynamic programming algorithms".

A Digital Humanist has to deal with trees in daily tasks. For instance, marking a text up using a specific markup language, i.e. a language for associating particular roles to the various parts of a text, is an activity one must address several times. Trees (as data structures) are the grounds for such a marking activity.

Marking up a text is something that, even implicitly, we do when we look at a piece of text, such as a novel. For instance, please consider the following excerpt from the first chapter of *Alice's Adventure in Wonderland* by Lewis Carroll [Carroll, 1866]:

> Alice was beginning to get very tired of sitting by her sister on the bank, and of having nothing to do: once or twice she had peeped into the book her sister was reading, but it had no pictures or conversations in it, "and what is the use of a book," thought Alice, "without pictures or conversations?"
>
> So she was considering in her mind, (as well as she could, for the hot day made her feel very sleepy and stupid,) whether the pleasure of making a daisy-chain would be worth

the trouble of getting up and picking the daisies, when suddenly a white rabbit with pink eyes ran close by her.

Each part of the text of the content above is organised precisely. Specific blocks of text are contained in paragraphs organised within chapters that finally compose the book.

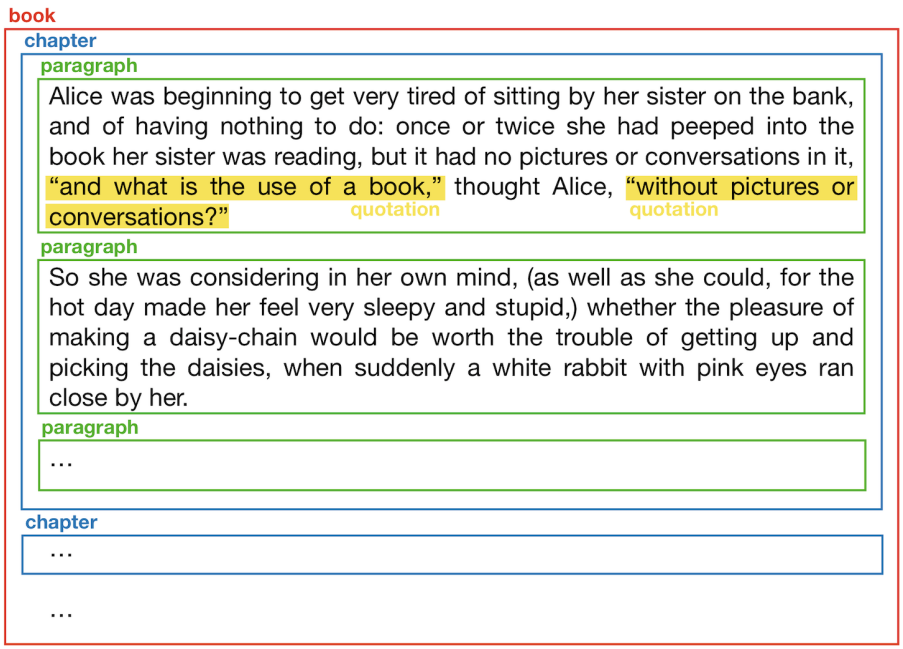

Figure 3. The first two paragraphs of *Alice's Adventure in Wonderland* marked up with basic textual structures.

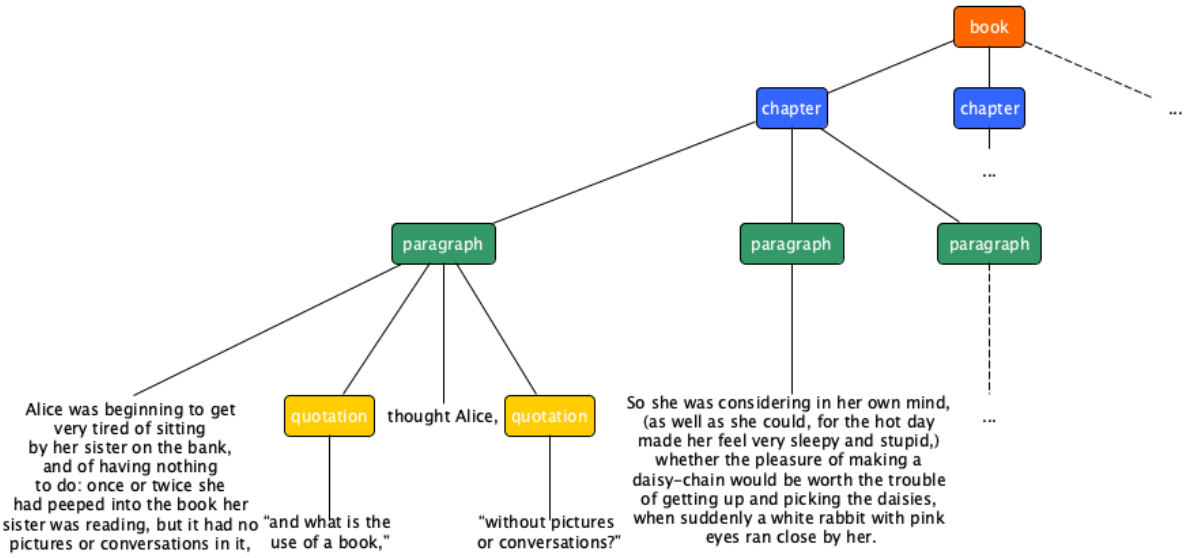

Figure 4. The tree that describes the containment of the various structures in the initial text of *Alice's Adventure in Wonderland*.

Besides, the text within each paragraph can contain additional structures, such as quotations when a character is speaking. Figure 3 shows all these structures. The main structure (i.e. *book*) is described as a box containing several boxes labelled *chapter*, each containing other boxes labelled *paragraph*, and so on. Enclosing part of a text within a labelled box is called markup. Appropriate markup languages have been defined to enable such annotations on a text.

```
<html>
    <head>
      <title>Alice's Adventures in Wonderland</title>
    </head>
    <body>
      <section role="doc-chapter">
            <p>
                Alice was beginning to get very tired of sitting by
                her sister on the bank, and of having nothing to do:
                once or twice she had peeped into the book her
                sister was reading, but it had no pictures or
                conversations in it, <q>and what is the use of a
                book,</q> thought Alice, <q>without pictures or
                conversations?</q>
            </p>
            <p>
                So she was considering in her own mind, (as well as
                she could, for the hot day made her feel very sleepy
                and stupid,) whether the pleasure of making a
                daisy-chain would be worth the trouble of getting up
                and picking the daisies, when suddenly a white
                rabbit with pink eyes ran close by her.
            </p>
            <p>...</p>
      </section>
      <section role="doc-chapter">...</section>
      ...
    </body>
</html>
```

Listing 1. A possible representation of the aforementioned text from *Alice's Adventures in Wonderland* in HTML.

The organisation mentioned above of boxes describes a clear hierarchy between them. For example, the bigger one (i.e. *book*) contains smaller ones (i.e. *chapters*), those include even smaller ones (i.e. *paragraphs*) and so on. When we are in the presence of such a hierarchical organisation of non-overlapping items, we can abstract it as a tree, as shown in Figure 4.

Languages such as the one provided by the Text Encoding Initiative (TEI) and the Hypertext Markup Language (HTML) are peculiar exemplars of markup languages. They allow one to construct hierarchies of markup elements for structurally and semantically annotating a text. For instance, Listing 1 shows a possible HTML representation of the quotation mentioned above from *Alice's Adventures in Wonderland*.

Trees

A *tree* is a data structure that simulates a hierarchical tree composed of a set of nodes related to each other by a particular hierarchical parent-child relation. As shown in Figure 5, this data structure usually follows a top-down presentation order, contrary to the actual real organisation of the plant.

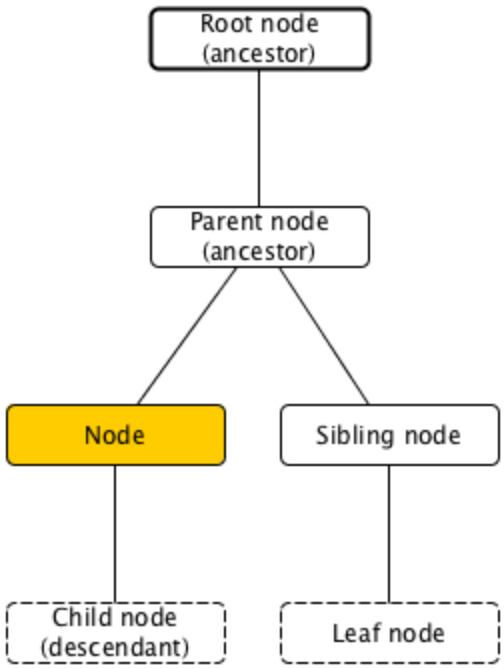

Figure 5. A tree with the typical jargon of its nodes considering the one highlighted in yellow as the focus. In this figure, the bold border identifies the tree's root node (i.e. the only node without any parent). Instead, the dashed border is used to indicate the leaf nodes of a tree (i.e. those nodes that do not have any children).

The originating (or starting) node is the *root node*, placed at the very top of the tree. Instead, the terminating nodes, called *leaf nodes*, are placed at the bottom. Considering a node of a tree, e.g. the yellow one highlighted in Figure 5, we can name precisely all the other nodes surrounding it. In particular:

- the *parent node* of a given one is a node directly connected to another one when moving close to the root node;
- a *child node* of a given one is a node directly connected to another one when moving away from the root node;
- a *sibling node* of a given one is a node that shares the same parent node;
- an *ancestor node* of a given one is a node that is reachable by following the child-parent path repeatedly;
- a *descendant node* of a given one is a node that is reachable by following the parent-child path repeatedly.

In Python, a tree is a set of nodes linked together according to parent-child relationships. Each tree is identified by its root node, which is unique. While there is no built-in implementation of the tree data structure in Python, some external packages are very well-suited for the task. Among those, one of the most famous ones is *anytree*.

This package allows one to create a tree node using the constructor `Node(name, parent=None)`. Thus, each node must specify a name that can be any Python object, such as a string, and a parent node. If the parent is not specified, it will assume *None* as value, which implicitly defines it as the tree's root node. In *anytree*, the constructor is the primary mechanism for determining a tree by merely stating the parent relations during the definition of new nodes.

It is worth mentioning that a parent node includes its children in a precise order. In particular, the order of insertion creates a kind of ordered list between all the sibling nodes. For instance, in the example in Listing 2, `paragraph_1` comes before `paragraph_2` in the siblings of the node body.

The advantages of using *anytree* are that it makes available many facilities for accessing various information associated with a node, using some variables associated with the class *Node*. The main variables are:

- `<node>.name` returns the object used as a name when creating `<node>`;
- `<node>.children` returns a tuple listing all the children of `<node>`;
- `<node>.parent` returns the parent of `<node>`;
- `<node>.descendants` returns a tuple listing all the descendants of `<node>` (including its children);
- `<node>.ancestors` returns a tuple listing all the ancestors of `<node>` (including its parent);
- `<node>.siblings` returns a tuple listing all the siblings of `<node>`;
- `<node>.root` returns the root node of the tree containing `<node>`.

```
from anytree import Node
```

```
book = Node("book")
chapter_1 = Node("chapter", book)
chapter_2 = Node("chapter", book)

paragraph_1 = Node("paragraph", chapter_1)
text_1 = Node("Alice was beginning to get very tired of sitting by "
           "her sister on the bank, and of having nothing to do: "
           "once or twice she had peeped into the book her sister "
           "was reading, but it had no pictures or conversations "
           "in it, ", paragraph_1)
quotation_1 = Node("quotation", paragraph_1)
text_2 = Node("“and what is the use of a book,”", quotation_1)
text_3 = Node(" thought Alice, ", paragraph_1)
quotation_2 = Node("quotation", paragraph_1)
text_4 = Node("“without pictures or conversations?”", quotation_2)

paragraph_2 = Node("paragraph", chapter_1)
text_5 = Node("So she was considering in her own mind, (as well as "
           "she could, for the hot day made her feel very sleepy "
           "and stupid,) whether the pleasure of making a "
           "daisy-chain would be worth the trouble of getting up "
           "and picking the daisies, when suddenly a white rabbit "
           "with pink eyes ran close by her.", paragraph_2)

paragraph_3 = Node("paragraph", chapter_1)
text_6 = Node("...", paragraph_3)
text_7 = Node("...", chapter_2)
text_8 = Node("...", book)
```

Listing 2. A simple tree depicting the textual structure introduced in Figure 4. The source code of this listing is available as part of the material of the course.

We can update the variables defining the children and the parent of a node by assigning to them a particular collection (e.g. a list or a tuple) of nodes. For instance, we can invert the ordering of the first two paragraphs defined as children of the first *chapter* node in Listing 2 as follows:

```
chapter_1.children = (paragraph_2, paragraph_1)
```

Also, it is possible to visualise the tree graphically on the shell by using appropriate tree *renderers*, described by the class *RenderTree* included in the *anytree* package. Once imported, one can create a new *RenderTree* object by specifying a node as input (e.g. the root node of the tree). Then one can print such a new renderer object to obtain a textual representation of the tree, as shown in the following excerpt:

```
from anytree import RenderTree


renderer = RenderTree(book)
print(renderer)

# Node('/book')
# ├── Node('/book/chapter')
# │   ├── Node('/book/chapter/paragraph')
# │   │   ├── Node('/book/chapter/paragraph/Alice was…')
# │   │   ├── Node('/book/chapter/paragraph/quotation')
# │   │   │   └── Node('/book/chapter/paragraph/quotation/“and…')
# │   │   ├── Node('/book/chapter/paragraph/ thought Alice, ')
# │   │   └── Node('/book/chapter/paragraph/quotation')
# │   │       └── Node('/book/chapter/paragraph/quotation/“without…')
# │   ├── Node('/book/chapter/paragraph')
# │   │   └── Node('/book/chapter/paragraph/So she was…')
# │   └── Node('/book/chapter/paragraph')
# │       └── Node('/book/chapter/paragraph/...')
# ├── Node('/book/chapter')
# │   └── Node('/book/chapter/...')
# └── Node('/book/...')
```

Exercises

1. Write in Python a *recursive* function `def breadth_first_visit(root_node)`. This function takes the root node of a tree and returns a list containing all the tree's nodes according to a breadth-first order. The breadth-first order considers all the nodes of the first level, then those of the second level, and so forth. For instance, considering the nodes created in Listing 2, the function called on the node `book` should return the following list: `[book, chapter_1, chapter_2, text_8, paragraph_1, paragraph_2, paragraph_3, text_7, text_1, quotation_1, text_3, quotation_2, text_5, text_6, text_2, text_4]`. Accompany the implementation of the function with the appropriate test cases.
2. Write the pure iterative version of the function defined in the previous exercise in Python.

Acknowledgements

The author wants to thank one of the students of the Digital Humanities and Digital Knowledge second-cycle degree of the University of Bologna – Severin Josef Burg – for having suggested corrections and improvements to the text of this chapter.

References

García Márquez, G. (1967). Cent'anni di solitudine. Mondadori, edizione 2017. ISBN: 978-8804675983

Carroll, L. (1866). Alice's Adventures in Wonderland. Macmillan and Co. Available at https://en.wikisource.org/wiki/Alice%27s_Adventures_in_Wonderland_(1866) (last visited 1 December 2019)