Organising information: trees

Author(s)
Silvio Peroni – silvio.peroni@unibo.it
Department of Classical Philology and Italian Studies, University of Bologna, Bologna, Italy

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

Abstract

These lecture notes introduce a new data structure for defining hierarchical relations: the tree. The historic hero introduced in these notes is Gabriel García Márquez, who was one of the most notable writers in Spanish of the 20th century. One of his novel, i.e. *One Hundred Years of Solitude*, is used here to introduce the way trees (as data structure) can be useful also to understand a story, or to structure a text.

Historic hero: Gabriel García Márquez

Gabriel García Márquez (shown in Figure 1) was a Colombian novelist, and he was one the most notable writers in Spanish of the 20th century. He won the Nobel Prize for Literature in 1982. While, as a journalist, he wrote several non-fictional works, 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, which is the main setting of one of his book, 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 generation of people of the same family, and follows its adventures and, often, misfortunes.

In the Italian edition of this novel, published by Mondadori [García Márquez, 1967], at the very beginning of the book is depicted a family tree of the various generations. While it provides a few spoilers about some future events, it is actually very useful to follow the story of the family, since several Buendia people often share the same name, and it is not so easy to follow the story if it is not clear to whom the narrator is actually referring to.

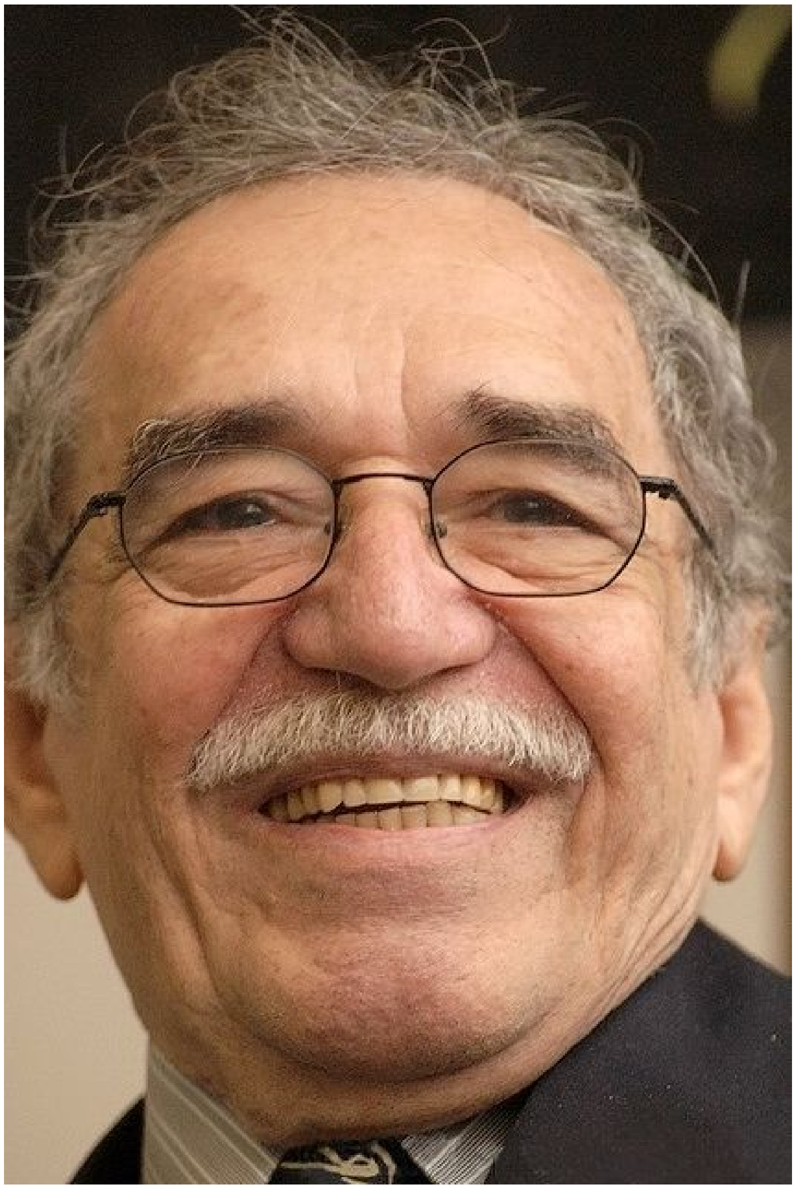


Figure 1. A portrait of Gabriel García Márquez. Picture by Jose Lara, source: https://en.wikipedia.org/wiki/File:Gabriel_Garcia_Marquez.jpg.

García Márquez is the second Latin American writer we have used in this course in order to introduce specific topics related to the Computational Thinking and Computer Science domains. In fact, trees (such as family trees) are particular kinds of structures that allow one to define a hierarchy of values that can be used for further tasks or computations.

Where is the tree?

Actually, it is not the first time we have adopted a tree for describing something, even if it was not mentioned explicitly. In particular, in the lecture "Dynamic programming algorithms", we used a tree (reprised in Figure 2) for showing the execution of the recursive calls to the function implementing the algorithm to find the Fibonacci number by means of the divide and conquer approach.

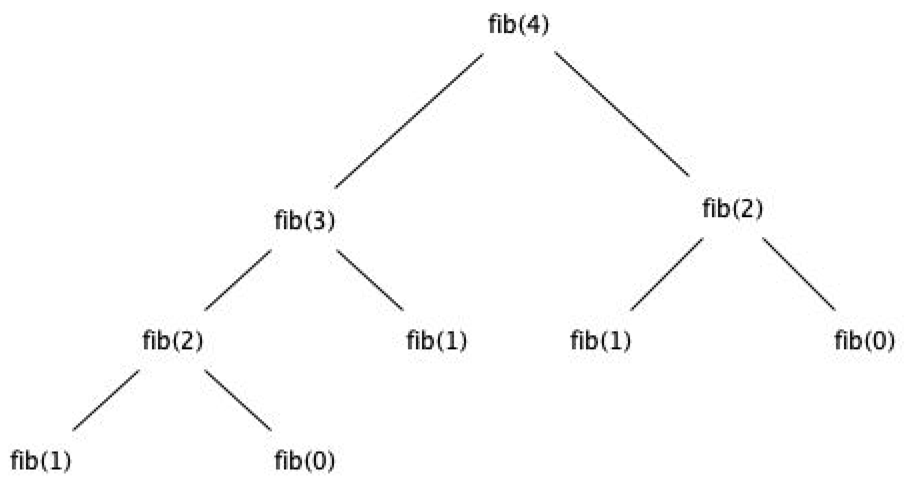
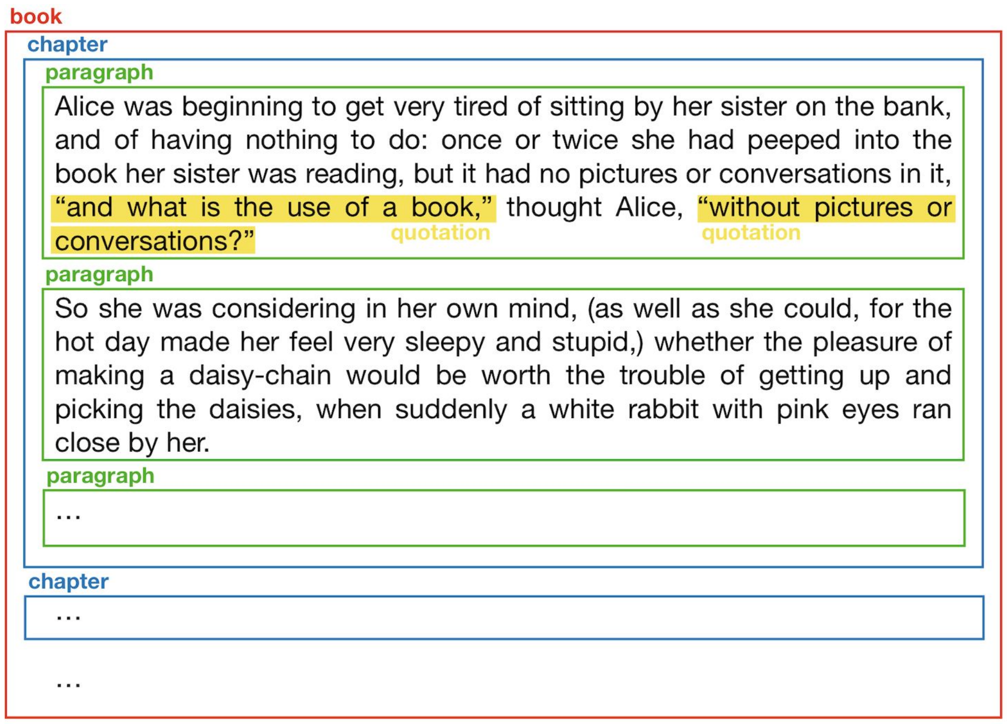


Figure 2. The tree describing the various recursive calls for calculating the Fibonacci number at the 4th month, as described in lecture "Dynamic programming algorithms".

In addition, as Digital Humanists, you will have to deal (or are already dealing) with trees when you have to marking a text up by means of a specific markup language – which is a language (again) for associating specific roles to the various parts of a text. This 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 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.

Even if it is totally implicit, each part of text of the aforementioned content is actually organised in a very precise way. In fact, specific blocks of text are contained in paragraphs, that are organised within chapters, that finally compose the book.



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?"

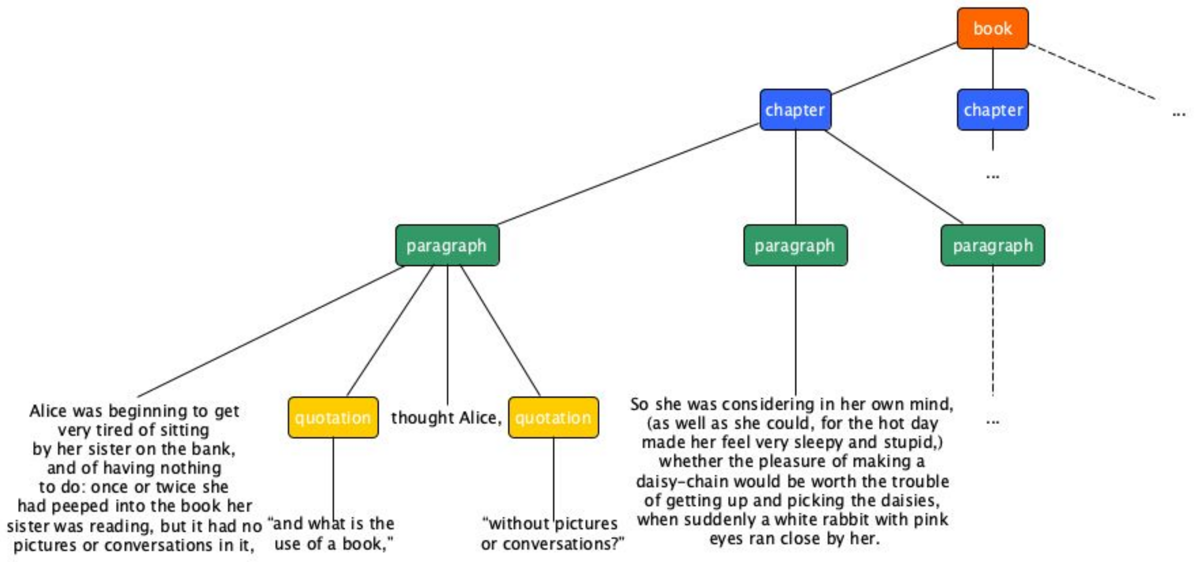
So she was considering in her own mind, (as well as she could, for the hot day made her feel very sleepy and stupid,) whether the pleasure of making a daisy-chain would be worth the trouble of getting up and picking the daisies, when suddenly a white rabbit with pink eyes ran close by her.

…

…

…

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



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,"

"without pictures or conversations?"

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.

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

In addition, the text within each paragraph can contain additional structures, such as quotations when a character is speaking. All these structure are explicitly shown in Figure 3, where the main structure (i.e. *book*) is described as a sort of box containing several boxes labelled *chapter*, each containing other boxes labelled *paragraph*, and so on. This approach of enclosing part of a text within a labelled box is called *markup*, and appropriate markup languages have been defined in the past so as to enable such kinds of 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.

Even if it is not extremely clear at a first sight, the aforementioned organisation of boxes describes a precise hierarchy between them, where the bigger one (i.e. *book*) contains smaller ones (i.e. *chapters*), those ones contain even smaller ones (i.e. *paragraphs*) and so on. When we are in presence of such hierarchical organisation of non-overlapping items, we can actually 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 which allows one to construct hierarchies of markup elements for structurally and semantically annotating a text. For instance, a possible HTML representation of the aforementioned quotation from *Alice's Adventures in Wonderland* is shown in Listing 1.

Trees

A *tree* is a data structure that simulates a hierarchical tree, composed by 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.

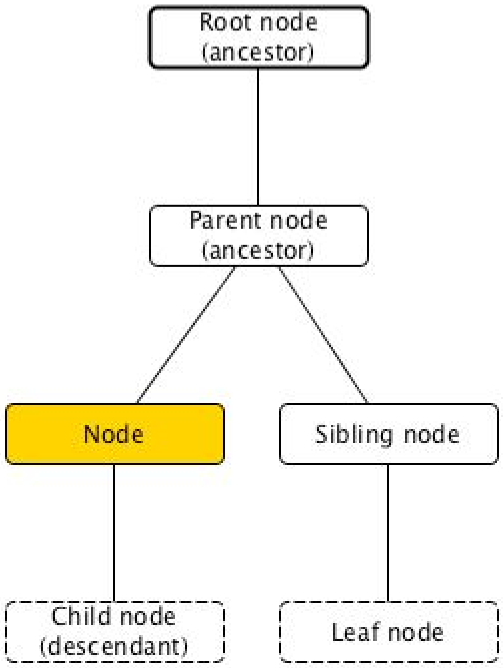


Figure 5. A tree with the typical nomenclature of its nodes considering the one highlighted in yellow as the focus. In this figure, the bold border is used to identify the root node of the tree (i.e. the only node without any parent), while a dashed border is used to indicate the leaf nodes of a tree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

The originating (or starting) node is called *root node* and is usually placed at the very top of the tree. Instead, the terminating nodes, called *leaf nodes*, are actually placed at the very bottom.

Taking into consideration 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 basically 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, there are some external packages that are very well-suited for the task. Among those, one of the most famous one is *anytree*.

This package allows one to create a node of a tree by means of 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, then it will assume *None* as value and it is implicitly defined as the root node of a tree. In *anytree*, the constructor is the main mechanism for defining a tree by simply stating the parent relations during the definition of new nodes.

It is worth mentioning that the siblings of a certain parent are actually ordered among them. The order is defined by the order of insertion as a child of that particular parent. 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 a lot of facilities for accessing various information associated to a node, by means of some variables associated to the class *Node*. The main variables are:

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

```
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.

It is worth mentioning that the variable defining the children and the parent of a node can be updated 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 firsts *chapter* node in Listing 2 as follows:

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

In addition, it is possible also to visualise the tree graphically on the shell, by using appropriate tree *renderers*. They are 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 three), and then one can print such new renderer object so as to have 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)` that takes the root node of a tree and returns a list containing all the nodes of the tree according to a breadth first order, which first consider all the nodes of the first level, then those ones 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 in Python the pure iterative version of the function defined in the previous exercise.

Acknowledgements

I would like to thank a student of the course, Severin Josef Burg, for having suggested corrections and improvements to the text of these lecture notes.

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 8 December 2018)