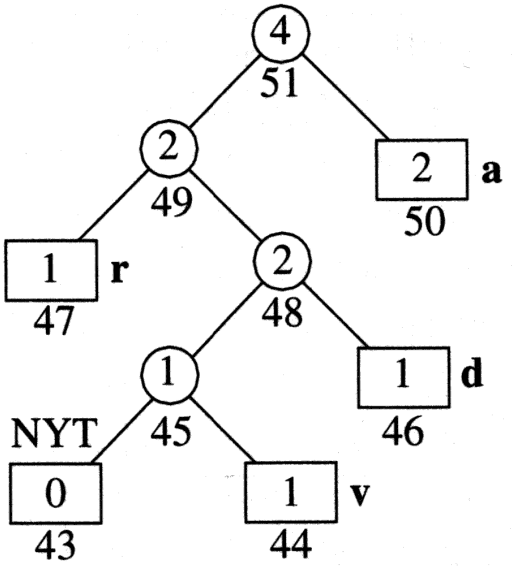
Trees

Carlos Moreno
cmoreno@uwaterloo.ca
EIT-4103



https://ece.uwaterloo.ca/~cmoreno/ece250

These slides, the course material, and course web site are based on work by Douglas W. Harder

Trees

Standard reminder to set phones to silent/vibrate mode, please!

Announcements

- Part of assignment 3 posted — additional questions on trees will be posted by Friday or possibly Monday. But you can get started with the questions on topics already covered.

Announcements

- Part of assignment 3 posted — additional questions on trees will be posted by Friday or possibly Monday. But you can get started with the questions on topics already covered.
- From what I'm told, Friday next week we can move our lecture to 3:30 (if everyone agrees).

Trees

- Today's class:
  - Definition of the tree data structure and its components
  - Related concepts:
    - Types of nodes, relations between nodes, paths and their attributes, subtrees
  - We'll look at one example — HTML/CSS as a hierarchical structure that can be represented with trees.

Trees

- As already mentioned earlier in the course, trees are the structure to be used to represent hierarchical data.

Trees

- As already mentioned earlier in the course, trees are the structure to be used to represent hierarchical data.
  - Perhaps a slight twist is that we will see some cases in which some characteristics of the tree will make them suitable for other situations as well, where the data is not hierarchical at all.

Trees

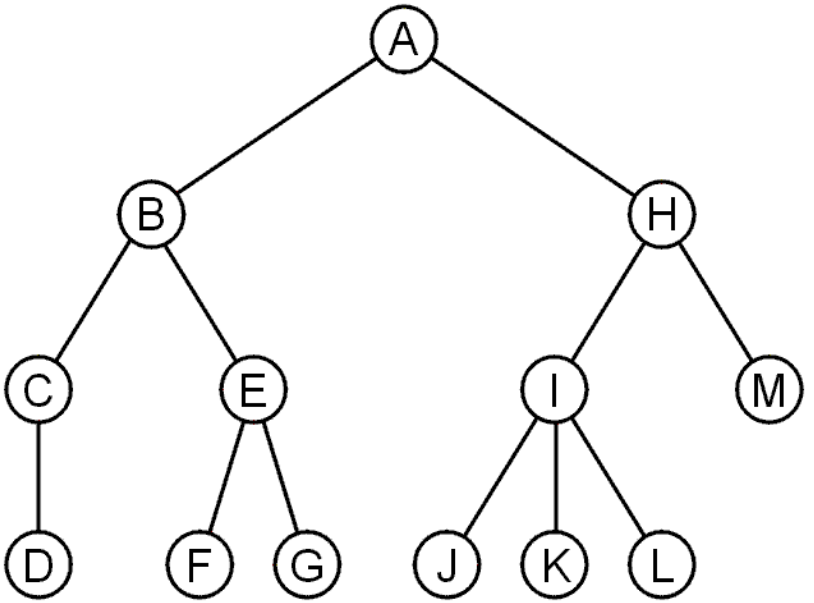
- A rooted tree data structure stores information in nodes
  - If we look at it by analogy with linked lists:
    - There is a first node, or root
    - Each node has variable number of next pointers
    - Each node, other than the root, has exactly one node pointing to it

Trees

- A rooted tree data structure stores information in nodes
  - If we look at it by analogy with linked lists:
    - There is a first node, or root
    - Each node has variable number of next pointers
    - Each node, other than the root, has exactly one node pointing to it
  - If we extend the analogy to doubly-linked lists:
    - Each node could have a “previous” pointer, pointing to the node immediately above it in the hierarchy.

Trees

- A rooted tree data structure stores information in nodes



Trees

- Terminology:
  - Parent and child nodes:
    - Respectively, the node immediately above and nodes immediately below in the hierarchy

Trees

- Terminology:
  - Parent and child nodes:
    - Respectively, the node immediately above and nodes immediately below in the hierarchy
  - Every tree has exactly one node that has no parent (*why just one?*)

Trees

- Terminology:
  - Parent and child nodes:
    - Respectively, the node immediately above and nodes immediately below in the hierarchy
  - Every tree has exactly one node that has no parent (*why just one?* — we recall that this is one of the aspects that distinguishes hierarchical from partial order; more than one can happen for partial orderings, but not for hierarchical orderings)

Trees

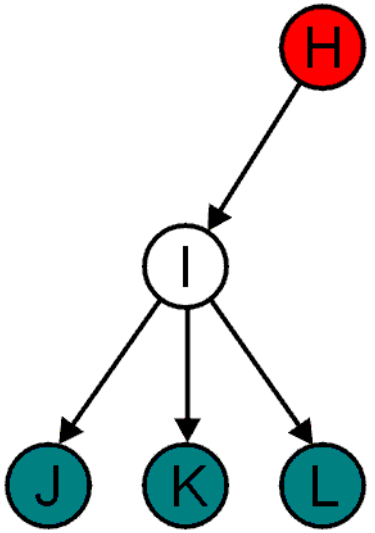
- Terminology:
  - Parent and child nodes:
    - Respectively, the node immediately above and nodes immediately below in the hierarchy
  - Every tree has exactly one node that has no parent (the top of the hierarchy)
    - This node is called the *root* node.

Trees

- Terminology:
  - Parent and child nodes:
    - Respectively, the node immediately above and nodes immediately below in the hierarchy
  - Every tree has exactly one node that has no parent (the top of the hierarchy)
    - This node is called the *root* node.
    - Every other node has:
      - Exactly one parent node, or just *parent*.
      - Zero or more child nodes, or just *children*.

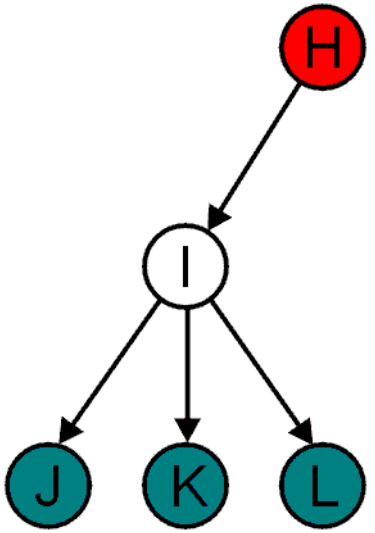
Trees

- Example:
  - Node H is the root node (the node containing H)
  - Node I has three children, and its parent is node H



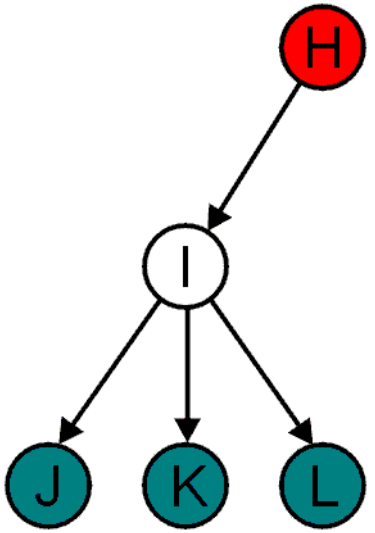
Trees

- Terminology:
  - The degree of a node is defined as the number of children it has — example: deg(I) is 3.



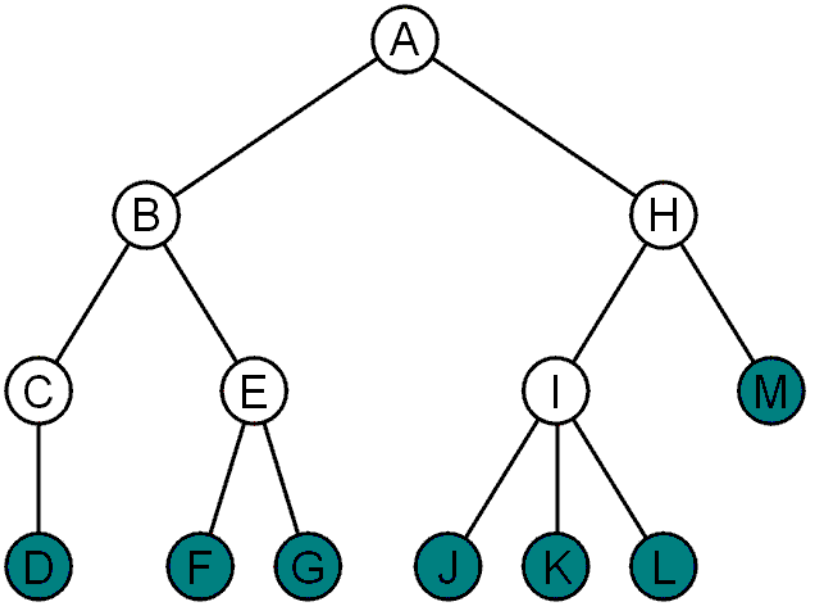
Trees

- Terminology:
  - Nodes with the same parent are *sibling nodes*, or just *siblings* — example: nodes J, K, L are siblings.



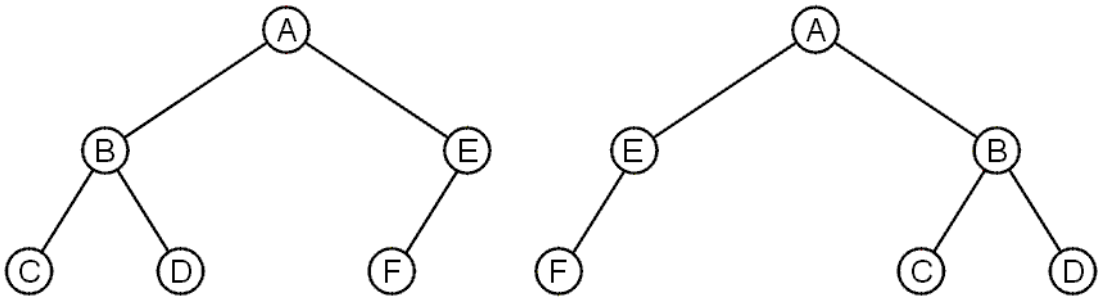
Trees

- Terminology:
  - Nodes with degree zero are called *leaf nodes*.
  - The others are called *internal nodes* (or informally, non-leaf nodes)



Trees

- Terminology:
  - These trees are equal if the order of the children is ignored (*unordered* trees)
  - They are different if order is relevant (*ordered* trees)

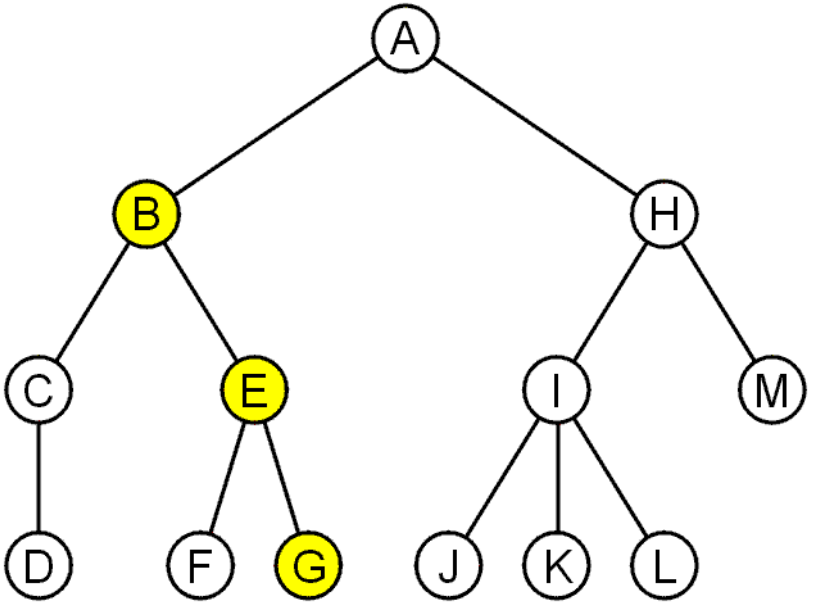


Trees

- Terminology:
  - A path is a sequence of nodes $(n_0, n_1, \cdots, n_L)$ where n_{k+1} is a child of n_k $\forall\ 0 \le k < L$
    - The *length* of this path is L — notice that the length is NOT the number of nodes involved; a good analogy would be that of the measured distance between the extremes.

Trees

- Example:
  - The path (B, E, G) has length 2

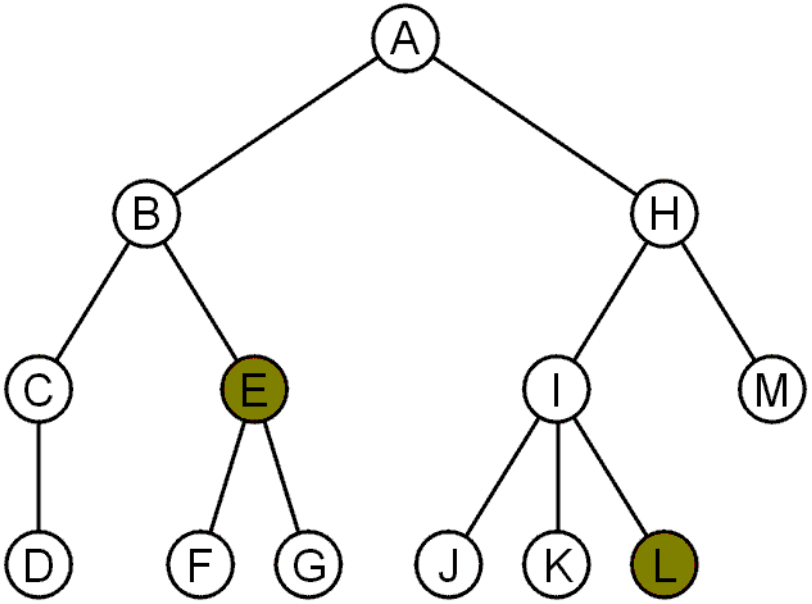


Trees

- For each node in a tree, there exists a *unique* path (*why unique*?) from the root to that node.

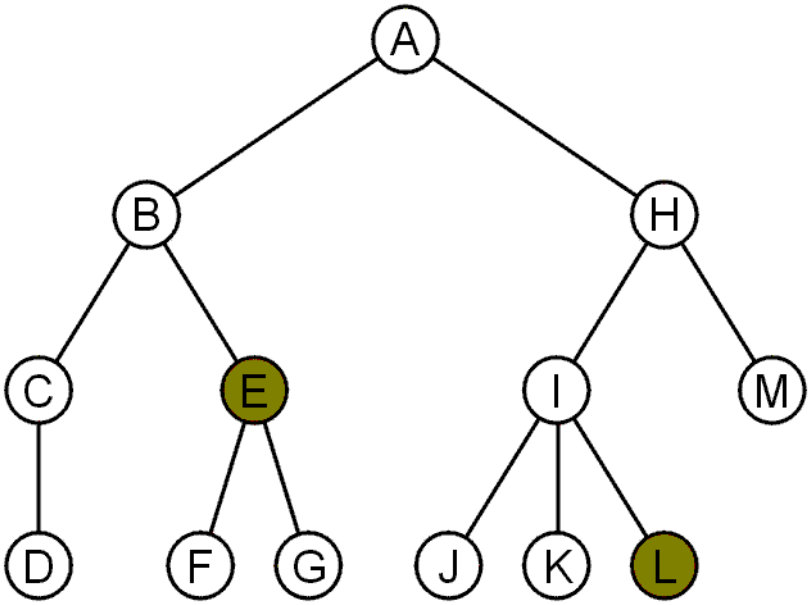
Trees

- For each node in a tree, there exists a *unique* path (*why unique*?) from the root to that node.
  - The length of this path is the *depth* of the node
    - Node E has depth 2
    - Node L has depth 3



Trees

- For each node in a tree, there exists a *unique* path (*why unique*?) from the root to that node.
  - The length of this path is the *depth* of the node
    - Node E has depth 2
    - Node L has depth 3
  - The root node has depth 0 (right?)



Trees

- The height of a tree is defined as the maximum depth of any node within the tree
  - The height of a tree with one node is 0
- For convenience, we define the height of an empty tree to be −1

Trees

- Terminology:
  - If a path exists from node *a* to node *b*, then:
    - *a* is an ancestor of *b*
    - *b* is a descendent of *a*

Trees

- Terminology:
  - If a path exists from node *a* to node *b*, then:
    - *a* is an ancestor of *b*
    - *b* is a descendent of *a*
  - Thus, every node is both an ancestor and a descendant of itself

Trees

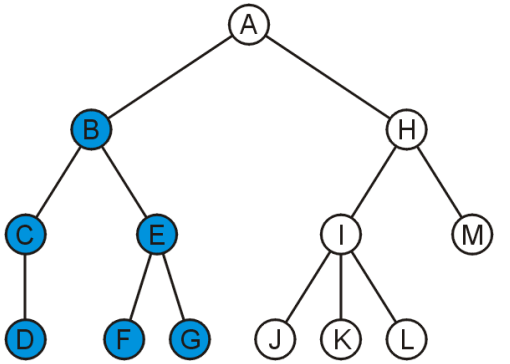
- Terminology:
  - If a path exists from node *a* to node *b*, then:
    - *a* is an ancestor of *b*
    - *b* is a descendent of *a*
  - Thus, every node is both an ancestor and a descendant of itself
  - We can add the qualifier *strict* to exclude equality: *a* is a *strict descendant* of *b* if *a* is a descendant of *b* but $a \neq b$

Trees

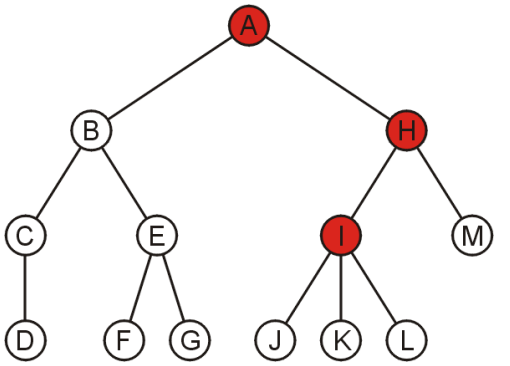
- Terminology:
  - If a path exists from node *a* to node *b*, then:
    - *a* is an ancestor of *b*
    - *b* is a descendent of *a*
  - Thus, every node is both an ancestor and a descendant of itself
  - We can add the qualifier *strict* to exclude equality: *a* is a *strict descendant* of *b* if *a* is a descendant of *b* but $a \neq b$
  - The root node is an ancestor of all nodes

Trees

- Example:
  - The descendants of node B are B, C, D, E, F, and G:



  - The ancestors of node I are I, H, and A:



Trees

- Terminology:
  - Given a node *N* within a tree with root *R*, the collection of all of the descendants of *N* (along with the associations) is said to be a *subtree* of the given tree, with root node *N*.

Trees

- Terminology:
  - Given a node *N* within a tree with root *R*, the collection of all of the descendants of *N* (along with the associations) is said to be a *subtree* of the given tree, with root node *N*.
  - In that spirit, we could provide a recursive definition of a tree:
    - A single node is a tree (node with degree 0)
    - A node with degree *n* is the root of a tree if all of its n children are the root of disjoint trees (no common nodes with the other subtrees)

Trees – Example/Case-Study

- A somewhat obvious example of when Tree data structures may be useful is HTML (or XHTML, XML, whatever they're calling it these days :-))
  - Writing (or reading) HTML of course does not require explicit use of tree data structures.
  - But how about a program that needs to process it?
    - Obvious example: the rendering engine of a web browser. It needs to take into account the hierarchical structure of the code to be able to determine the actual graphical look (rendering).

Trees – Example/Case-Study

- A somewhat obvious example of when Tree data structures may be useful is HTML (or XHTML, XML, whatever they're calling it these days :-))
  - Even more: the document can be dynamically modified (for example, through JavaScript, which is a client-side, browser-powered scripting language)
    - So, the browser software better have efficient mechanisms to determine the new rendering!

Trees – Example/Case-Study

- A somewhat obvious example of when Tree data structures may be useful is HTML (or XHTML, XML, whatever they're calling it these days :-))
  - Again, we keep in mind: HTML/XML is hierarchical by nature, so a data structure that represents this hierarchy has to be the right solution!

Trees – Example/Case-Study

- The main idea (this is way oversimplified, BTW) with HTML is the use of tags that define a component (usually something that will display on the browser area).
  - Tags can enclose other (nested) tags.
    - Not unlike blocks in a program.
  - Tags usually have opening and closing components, enclosed in angle brackets (< >), with the closing one having a slash before the name:
    - Example: `<title>ECE-250</title>`

Trees – Example/Case-Study

- Consider a more general example:

```
<html>
    <head>
        <title>Hello World!</title>
    </head>
    <body>
        <h1>This is a <u>Heading</u></h1>

        <p>This is a paragraph with some
        <u>underlined</u> text.</p>
    </body>
</html>
```

Trees – Example/Case-Study

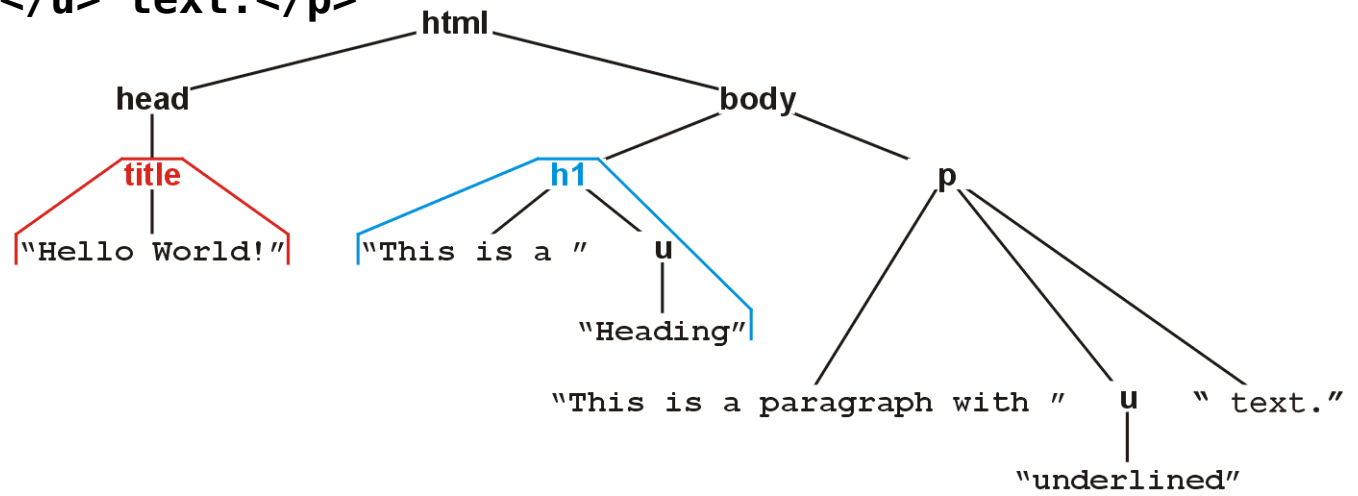
- The hierarchy of nested tags defines a tree with root being the **<html>** tag:

```
<html>
    <head>
        <title>Hello World!</title>
    </head>
    <body>
        <h1>This is a <u>Heading</u></h1>

        <p>This is a paragraph with some
        <u>underlined</u> text.</p>
    </body>
</html>
```

Trees – Example/Case-Study

- The hierarchy of nested tags defines a tree with root being the **<html>** tag:

```
<html>
    <head>
        <title>Hello World!</title>
    </head>
    <body>
        <h1>This is a <u>Heading</u></h1>

        <p>This is a paragraph with some
        <u>underlined</u> text.</p>
    </body>
</html>
```

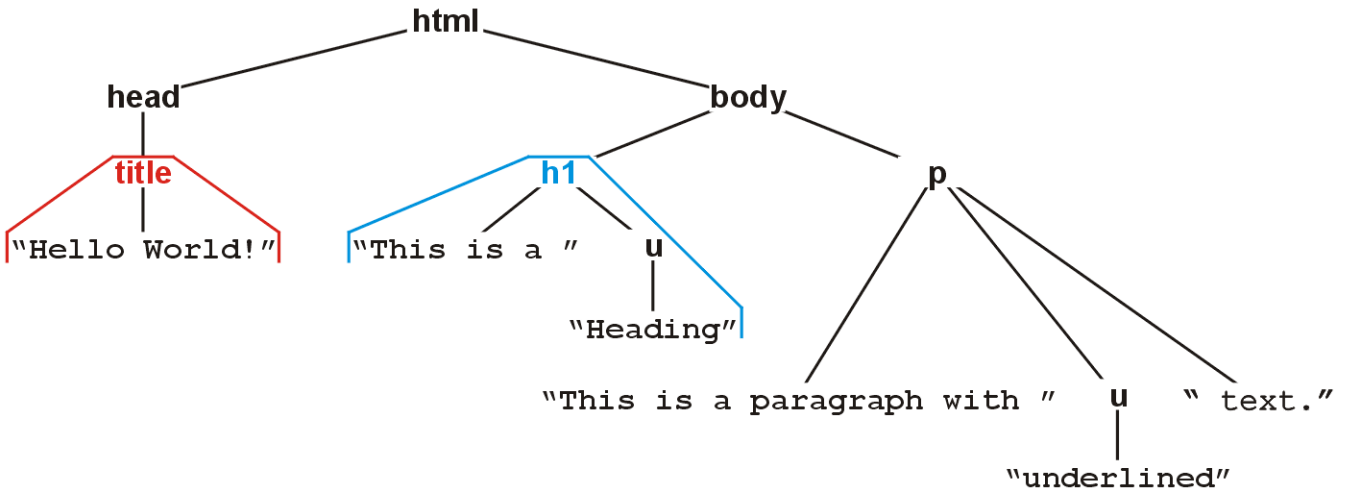


Trees – Example/Case-Study

- The rendering software defines a tree structure where the nodes store the attributes of the tag, and the child nodes are the nested tags.
- This allows it to efficiently propagate (inherit) attributes (font, size, color, border, etc.) to all nested tags (a behaviour specified by HTML)

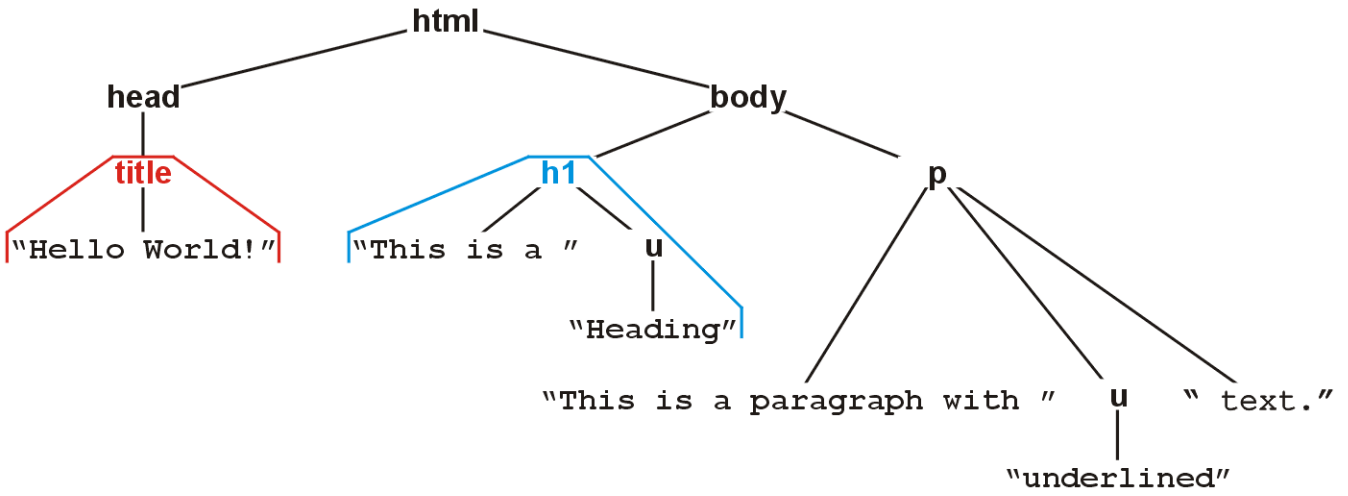
Trees – Example/Case-Study

- For example, if the <body> tag has some attributes associated, then the tree allows the browser software to efficiently determine where these attributes should propagate:



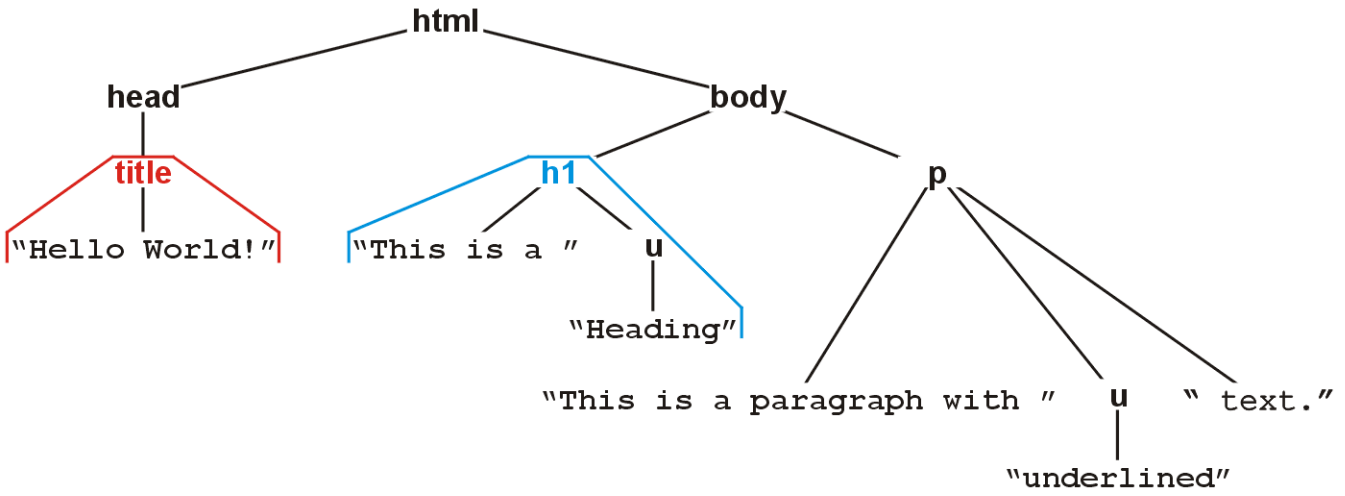
Trees – Example/Case-Study

- Perhaps even more important, when dynamically changing things, the browser can efficiently determine what elements are affected, just following links in the tree:



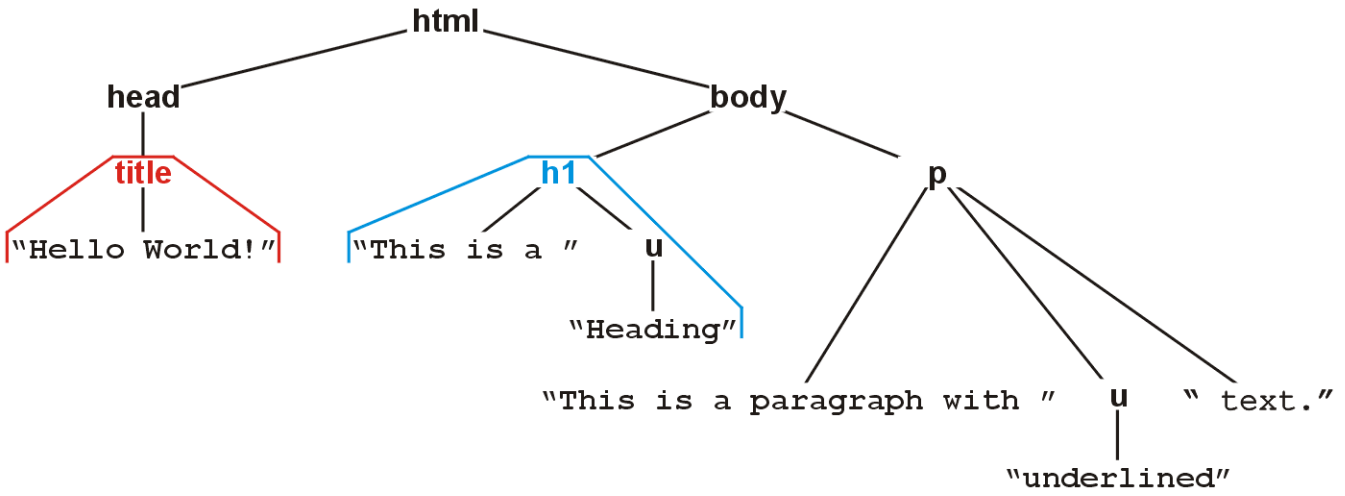
Trees – Example/Case-Study

- Notice that this includes the fact that children nodes will (may) have an effect on the parents, repetitively all the way up to the root:
  - If the font size of the second tag **<u>** is dynamically changed, we need be able to efficiently determine that this will make the **<p>** tag's rendering increase, making the **<body>** tag affected as well.



Trees – Example/Case-Study

- An observation: this is an example of an *ordered* tree. In HTML, tags are graphically rendered according to the order in which they appear (by default — if we use CSS to define absolute positioning, things change; still, that's usually done, if at all, just for some of the tags)



Trees

- Bottom line:
  - Trees are a very fundamental and very powerful data structure:
  - You *will* see them again in the near future (and I mean your future courses, not the future lessons in our course! :-))

Trees

- Bottom line:
  - Trees are a very fundamental and very powerful data structure:
  - You *will* see them again in the near future (and I mean your future courses, not the future lessons in our course! :-))
    - That you take a path in Control systems, Telecomm and Digital signal processing, Image processing or Computer graphics, embedded systems — no matter what, you'll be seeing heavy use of trees as a powerful tool to efficiently solve many of the domain-specific problems.

Trees

- Bottom line:
  - Trees are a very fundamental and very powerful data structure:
    - That you take a path in Control systems, Telecomm and Digital signal processing, Image processing or Computer graphics — no matter what, you'll be seeing heavy use of trees as a powerful tool to efficiently solve many of those problems.
    - HTML will seem like a toy example by comparison (we'll go over some of these additional case-studies as we progress with our coverage of trees)

Summary

- During today's class, we discussed:
  - Definition of trees and related components
    - Root, internal, and lead nodes.
    - Parent, children, and sibling nodes.
    - Path, path length, height, depth.
    - Ancestors and descendants.
    - Ordered and unordered trees.
    - Subtrees.
  - An example/case-study: HTML rendering.