CSC 212: Data Structures and Abstractions

14: Trees

Marco Alvarez

Department of Computer Science and Statistics
University of Rhode Island

Fall 2017

THINK BIG WE DO℠

Trees

Trees

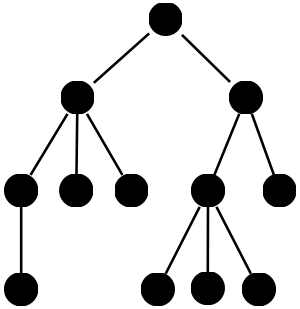
- List, Stacks, Queues are **linear data structures**

Trees

- List, Stacks, Queues are **linear data structures**
- Trees allow for **hierarchical** relationships
  - nodes have **parent-child** relation

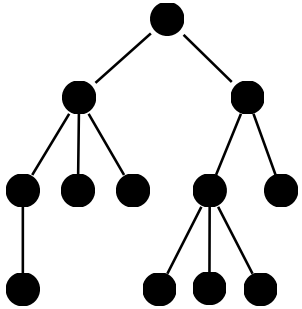
Trees

- List, Stacks, Queues are **linear data structures**
- Trees allow for **hierarchical** relationships
  - nodes have **parent-child** relation

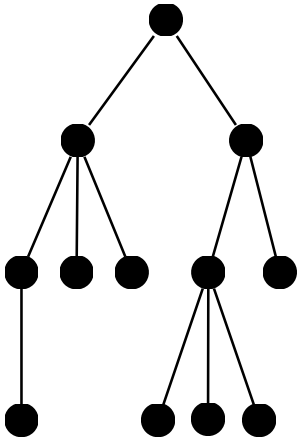


Trees

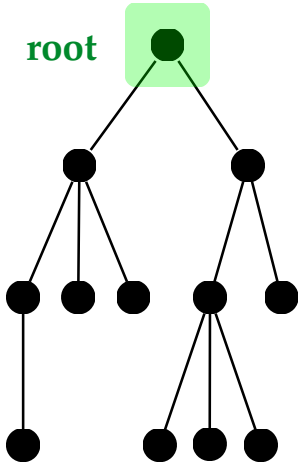
- List, Stacks, Queues are **linear data structures**
- Trees allow for **hierarchical** relationships
  - nodes have **parent-child** relation



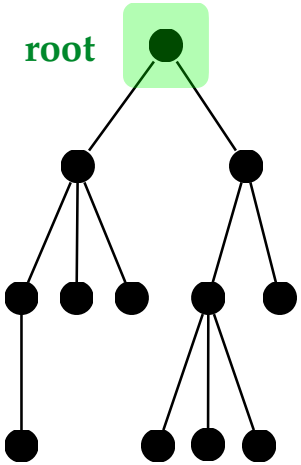
General Trees (definition)



General Trees (definition)

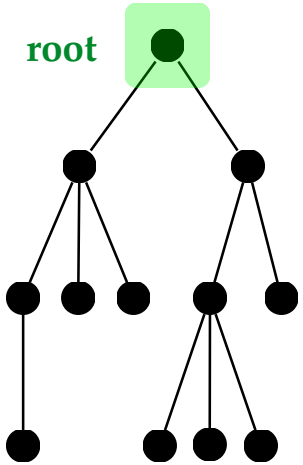


General Trees (definition)



There is a **unique path** from the root to each node in the tree

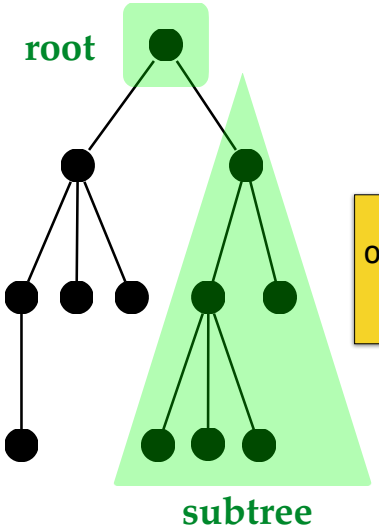
General Trees (definition)



There is a **unique path** from the root to each node in the tree

A **tree** is either **empty** or a root node connected to 0 or more trees (called **subtrees**)

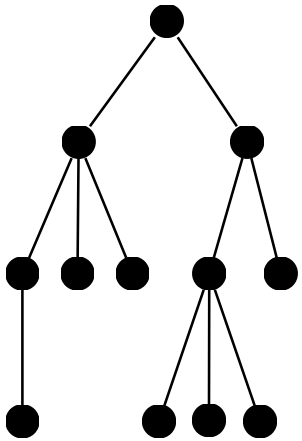
General Trees (definition)



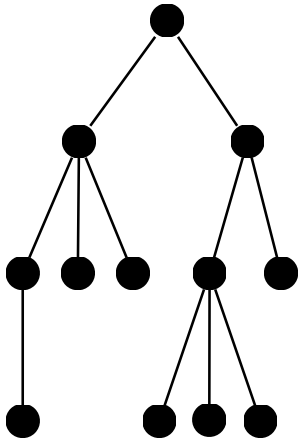
There is a **unique path** from the root to each node in the tree

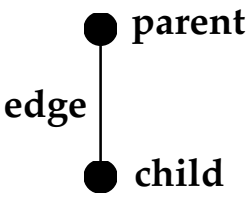
A **tree** is either **empty** or a root node connected to 0 or more trees (called **subtrees**)

Trees (jargon)

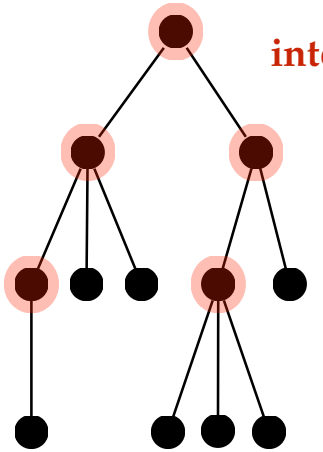


Trees (jargon)

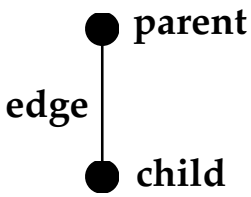




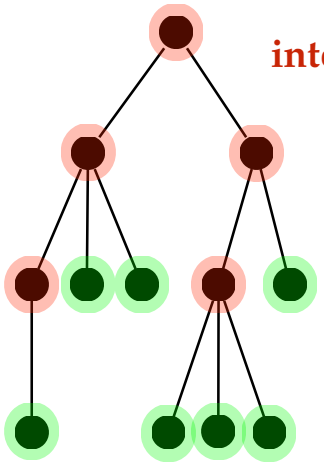
Trees (jargon)



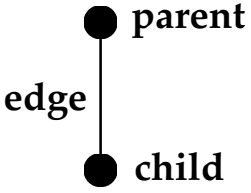
internal nodes



Trees (jargon)



internal nodes



leaves

Trees (jargon)

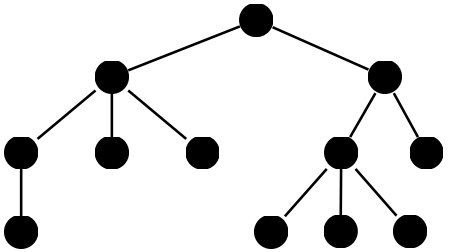
- Each node is either a **leaf** or an **internal node**
  - an internal node has one or more children
  - a leaf node (external node) has no children

Trees (jargon)

- Each node is either a **leaf** or an **internal node**
  - an internal node has one or more children
  - a leaf node (external node) has no children
- Nodes with the same parent are **siblings**

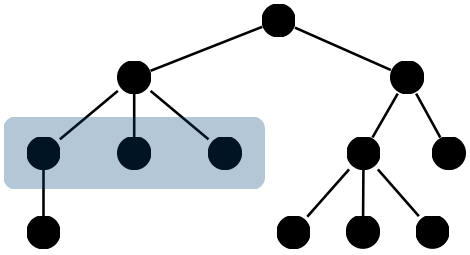
Trees (jargon)

- Each node is either a **leaf** or an **internal node**
  - an internal node has one or more children
  - a leaf node (external node) has no children
- Nodes with the same parent are **siblings**

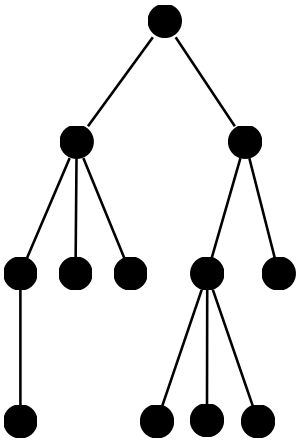


Trees (jargon)

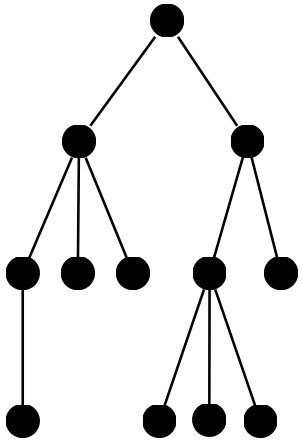
- Each node is either a **leaf** or an **internal node**
  - an internal node has one or more children
  - a leaf node (external node) has no children
- Nodes with the same parent are **siblings**



Paths

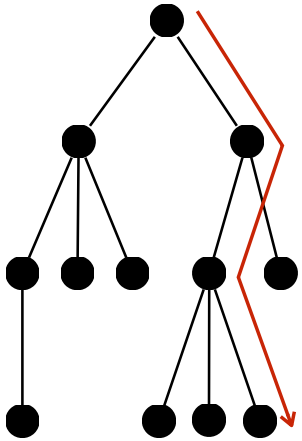


Paths



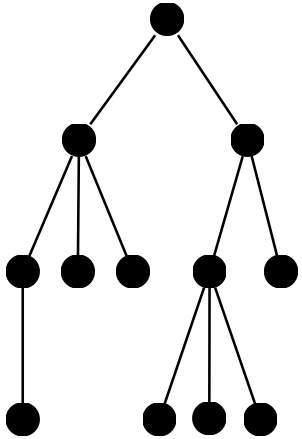
A **path** from node v_0 to v_n is a sequence of nodes $v_0, v_1, v_2, \ldots, v_n$, where there is an edge from one node to the next

Paths



A **path** from node v_0 to v_n is a sequence of nodes $v_0, v_1, v_2, \ldots, v_n$, where there is an edge from one node to the next

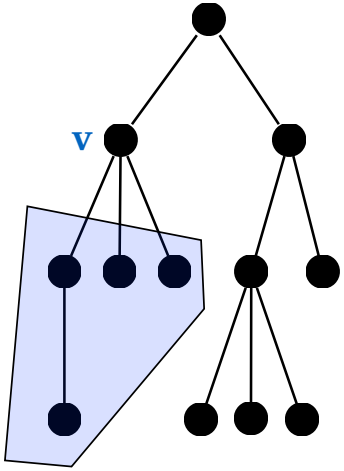
Paths



A **path** from node v_0 to v_n is a sequence of nodes $v_0, v_1, v_2, \ldots, v_n$, where there is an edge from one node to the next

The **descendants** of a node **v** are all nodes reached by a path from node **v** to the leaf nodes

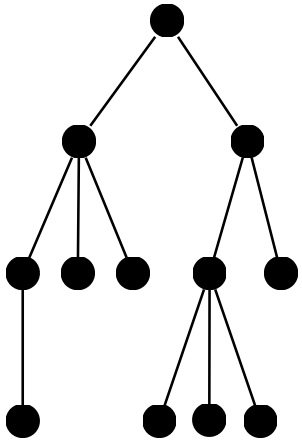
Paths



A **path** from node v_0 to v_n is a sequence of nodes $v_0, v_1, v_2, \ldots, v_n$, where there is an edge from one node to the next

The **descendants** of a node **v** are all nodes reached by a path from node **v** to the leaf nodes

Paths

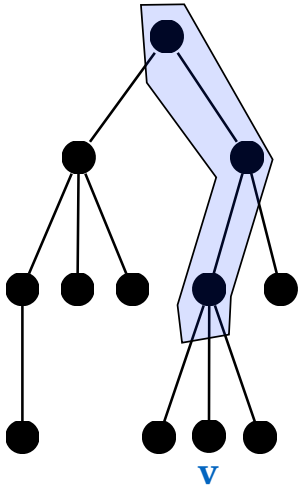


A **path** from node v_0 to v_n is a sequence of nodes $v_0, v_1, v_2, \ldots, v_n$, where there is an edge from one node to the next

The **descendants** of a node **v** are all nodes reached by a path from node **v** to the leaf nodes

The **ancestors** of a node **v** are all nodes found on the path from the root to node **v**

Paths

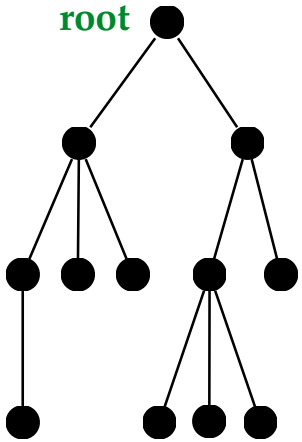


A **path** from node v_0 to v_n is a sequence of nodes $v_0, v_1, v_2, \ldots, v_n$, where there is an edge from one node to the next

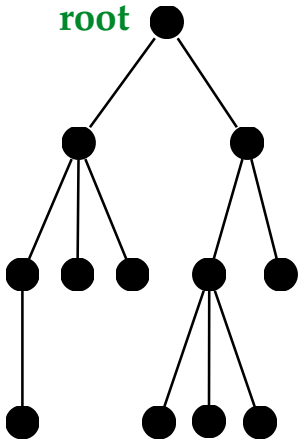
The **descendants** of a node **v** are all nodes reached by a path from node **v** to the leaf nodes

The **ancestors** of a node **v** are all nodes found on the path from the root to node **v**

Depth and Height

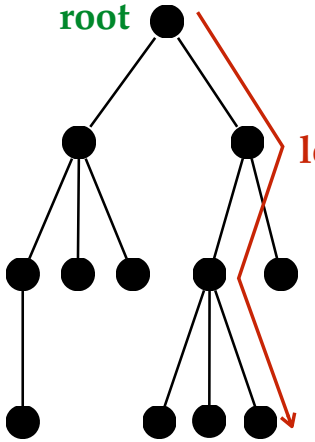


Depth and Height



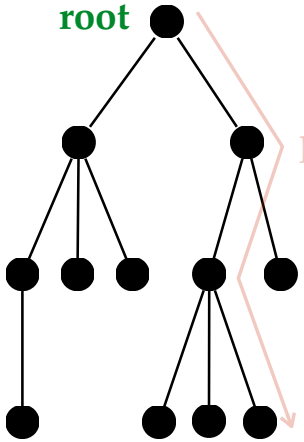
The length of a **path** is the number of edges in the path

Depth and Height



The length of a **path** is the number of edges in the path

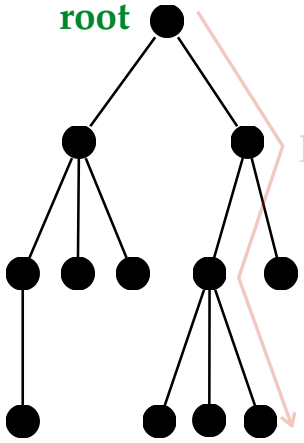
Depth and Height



The length of a **path** is the number of edges in the path

The **depth** (level) of a node **v** is the length of the path from the root node to **v**

Depth and Height

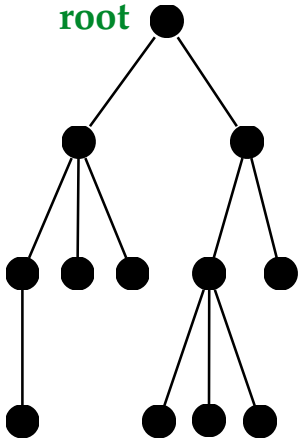


The length of a **path** is the number of edges in the path

The **depth** (level) of a node **v** is the length of the path from the root node to **v**

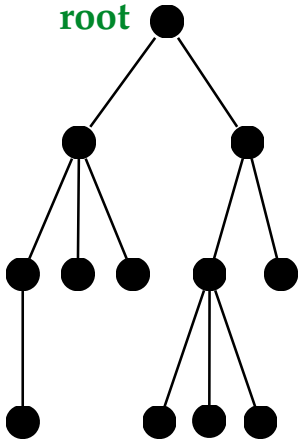
The **height** of a node **v** is the length of the path from **v** to its deepest descendant

Tree Properties



The **depth of the tree** is the depth of deepest node

Tree Properties



The **depth of the tree** is the depth of deepest node

The **height of the tree** is the height of the root

How to implement general trees?

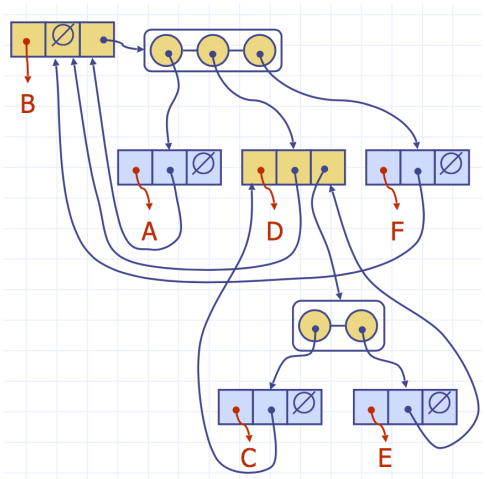
Node:

data
parent
children array

How to implement general trees?

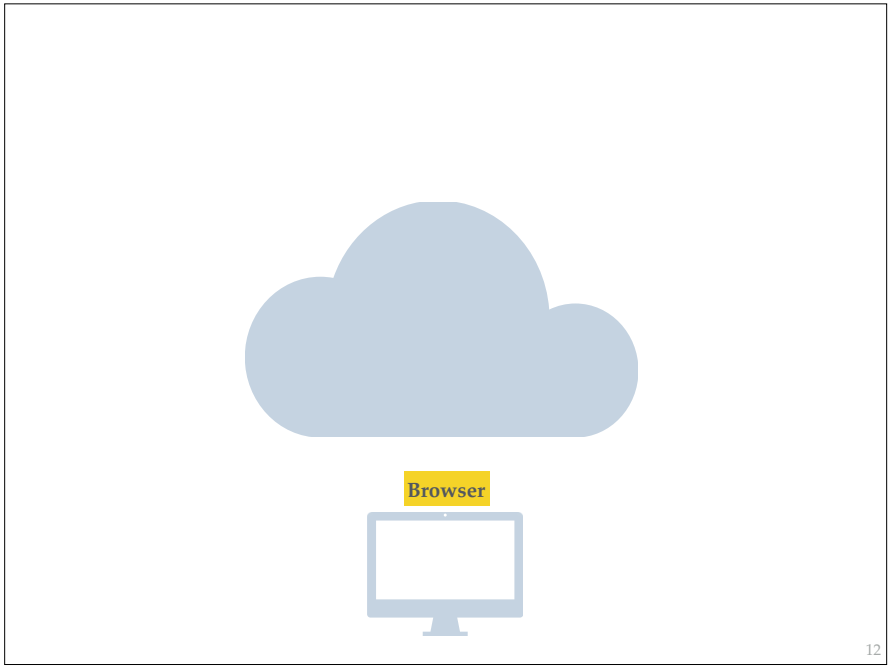
Node:

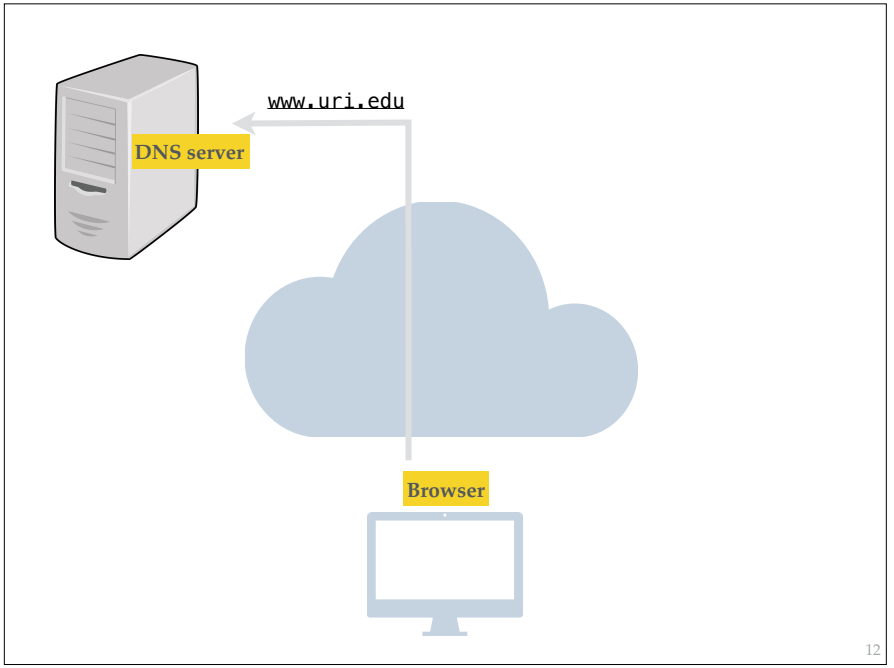
data
parent
children array

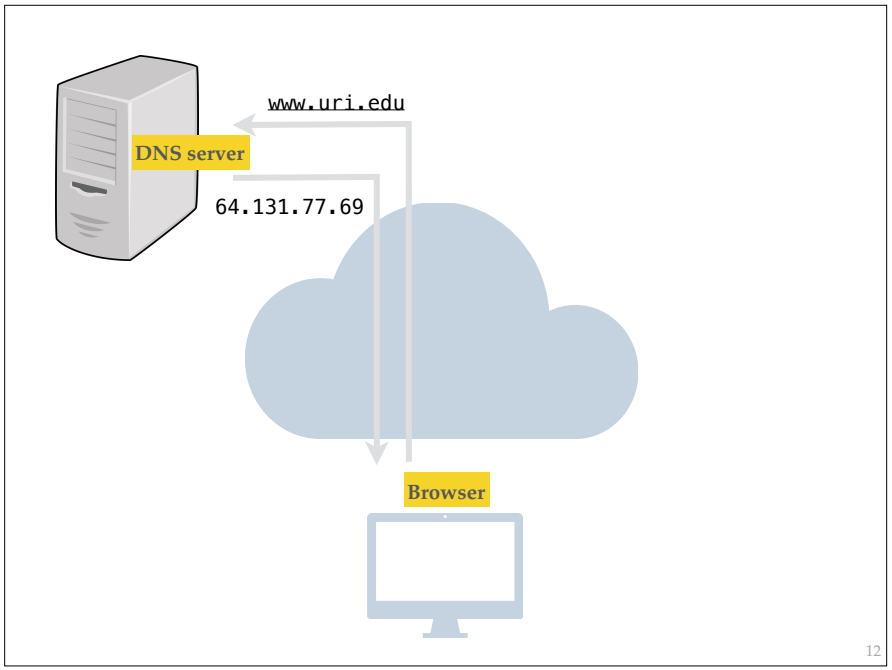


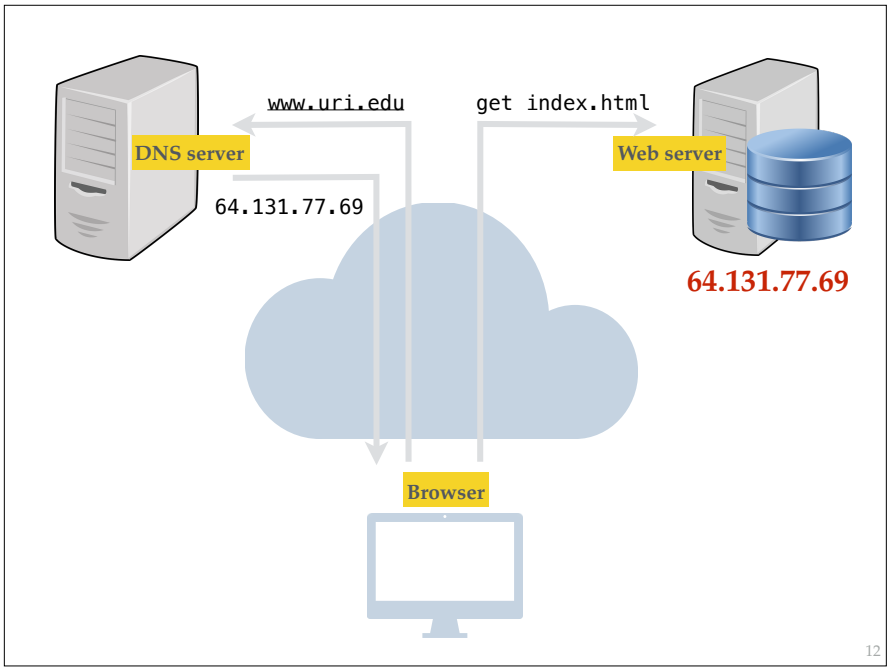
From Algorithm Design and Applications, Goodrich & Tamassia

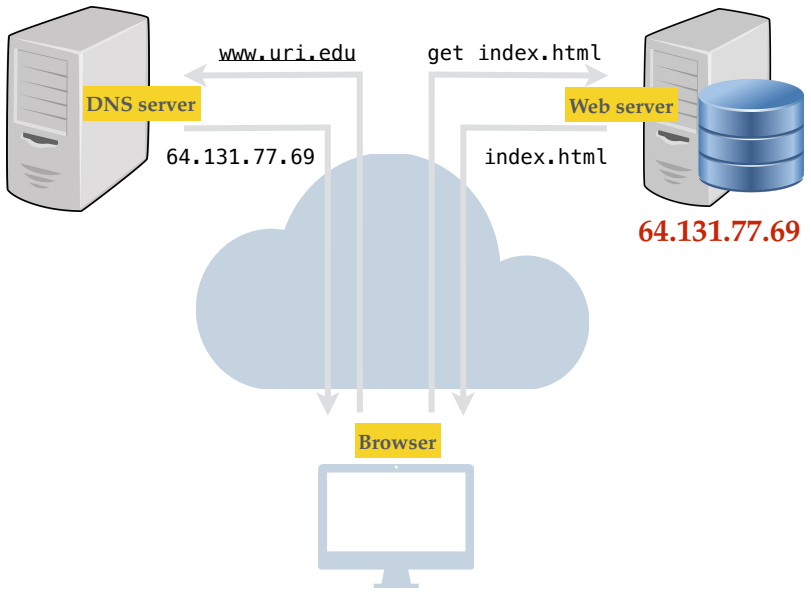
Traversals


Browser

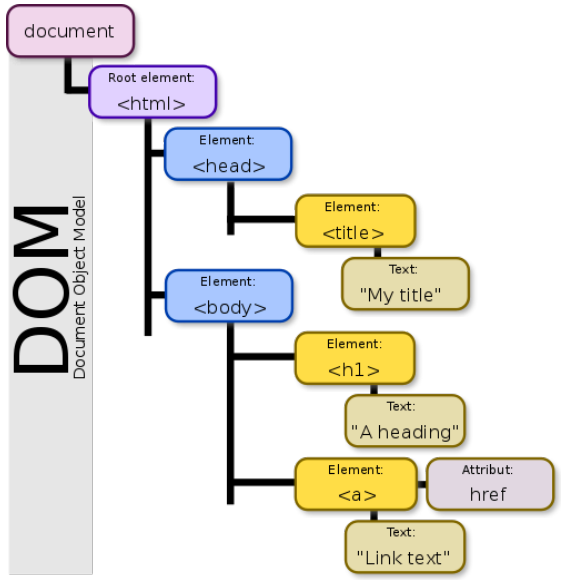

www.uri.edu
DNS server
Browser

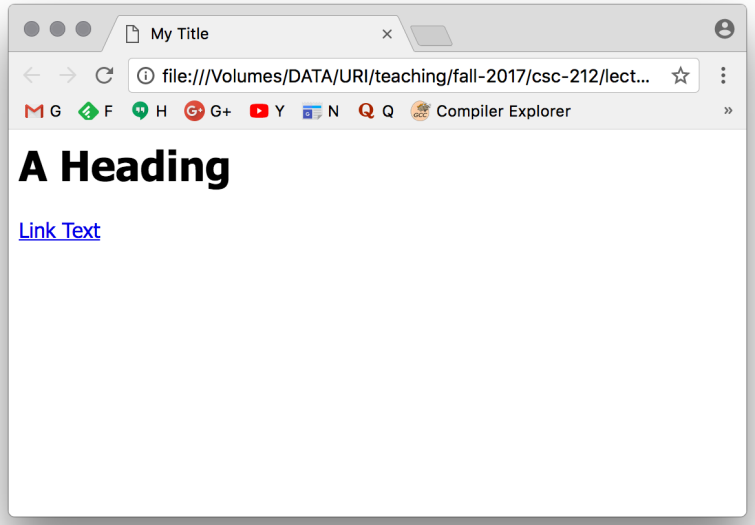

www.uri.edu
DNS server
64.131.77.69
Browser


www.uri.edu
get index.html
DNS server
Web server
64.131.77.69
64.131.77.69
Browser




```
 1 <html>
 2
 3 <head>
 4 <title>My Title</title>
 5 </head>
 6
 7 <body>
 8 <h1>A Heading</h1>
 9 <a href="http://www.uri.edu">Link Text</a>
10 </body>
11
12 </html>
```

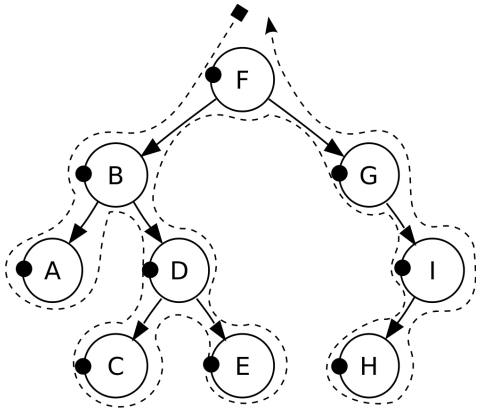





Traversing a tree

a traversal is a method that “**visits**” every node in a tree once

Traversing a tree

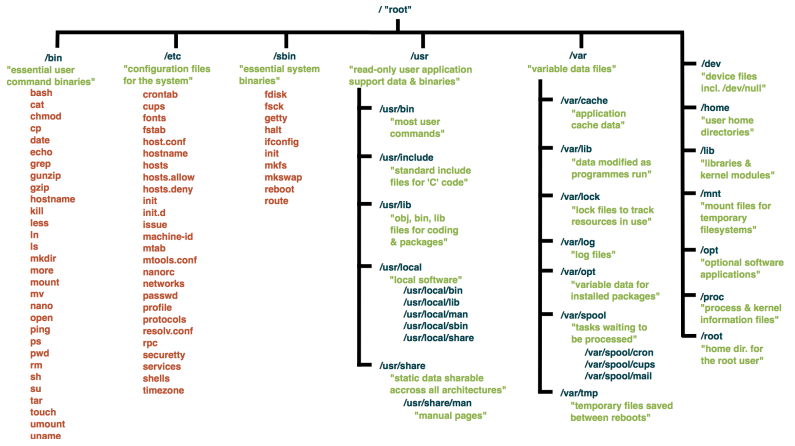


a traversal is a method that “**visits**” every node in a tree once

Preorder Traversal

```
1 algorithm preorder(p) {
2     visit(p)
3     for each child c of p {
4         preorder(c)
5     }
6 }
```


UNIX File System



How to compute amount of space used by files in folders and subfolders?

Postorder Traversal

```
1 algorithm postorder(p) {
2     for each child c of p {
3         postorder(c)
4     }
5     visit(p)
6 }
```