COMP20230: Data Structures & Algorithms
Lecture 15: Trees

Dr Andrew Hines

Office: E3.13 Science East
School of Computer Science
University College Dublin

andrew.hines@ucd.ie

Trees

Last Week

- **Sorting:** Different Algorithms to reorder data. Same input/output but different implementations and performance.

Still to see...

- **Trees:** Tree (ADT) and tree searching
- **Graphs:** Graph (ADT), Minimum Spanning Trees (Kruskal, Prim); Dijkstra's Shortest Path Algorithm

Pulling it all together: Algorithms and Data Structures

ADT	Stack, Queue, Sequence, Set, **Tree**, Graph
Type of data structure	**Array-based**, **Linked list**
Python data structures	List, String, Tuple
Algorithms	Sorting, search in graph, shortest path, minimum spanning tree

Outline

What is a tree?

- Definitions
- Tree ADT
- Array-based Representation
- Linked List-based Representation
- Traversing a tree: DFS and BFS

Take home message

Tree ADT is a *non-linear* data type

Linear vs. Non-Linear Data Structures

Linear data structure

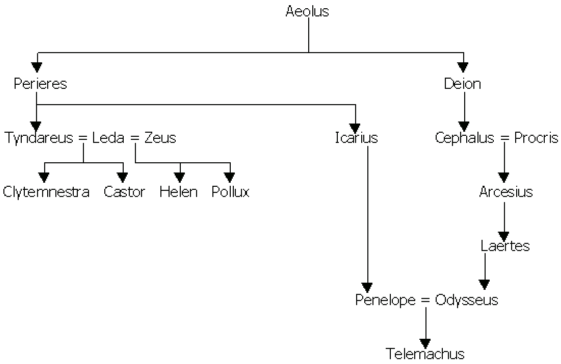
Data in an organised a linear fashion, i.e. in the form of a list.

Non-linear data structure

A tree is an example of a non-linear data structure. Tree has one root node that acts as a starting point and links to other nodes.

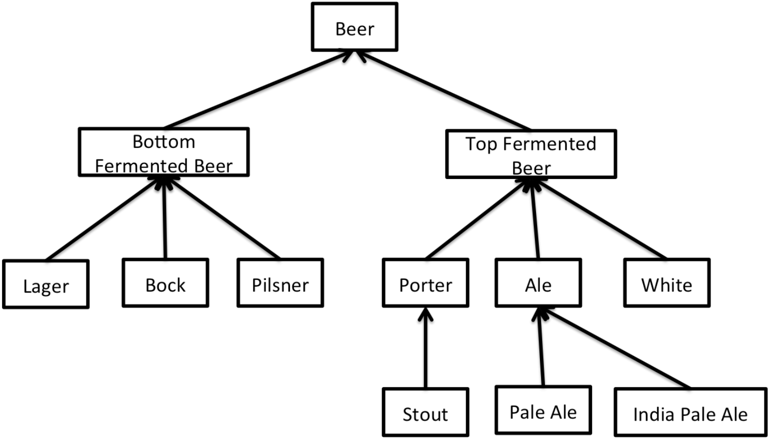
Example Tree (not a tree)

House of Troy (Greek Mythology)



While a family tree is a familiar tree, it is actually a directed acyclic graph (DAC) ADT as relatives can mate but cannot be there own ancestor.

Example Tree



Trees

Non-linear structures are very important in the IT industry

- file systems
- data base systems
- languages (programming) have a hierarchical structure
  $\Longrightarrow$ notion of Abstract Syntax Tree

Terminology

Relationship between elements is hierarchical:

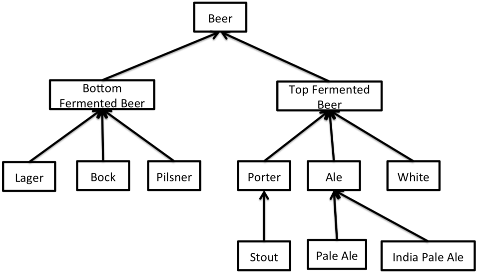
- root
- above/below
- parent/child
- ancestor/descendant

General Definition

Tree

An abstract data type that stores elements hierarchically.

- Each element has a **parent** element and zero or more **child** elements (children)
- The top element is called the **root** of the tree



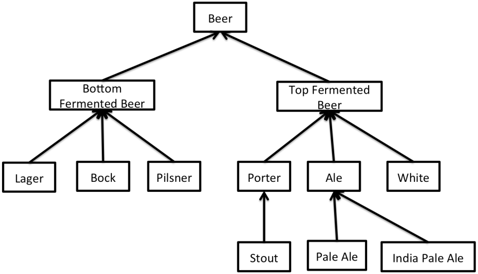
Tree: Formal Definition

A **tree**, `T`, is a set of **nodes** storing elements such that the nodes have a **parent-child** relationship that satisfies the following properties:

- If `T` is non-empty it has a special node, called the **root** of `T` , that has no parent
- Each node `v` of `T` different from the root has a unique **parent** node `w`; every node with parent `w` is a child of `w`

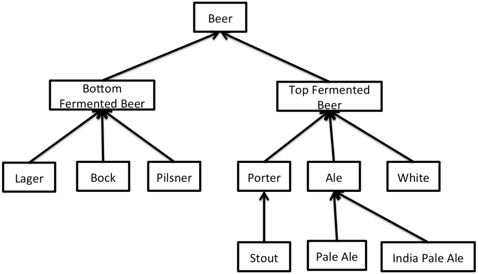
Other Node Relationships

- **Siblings**: two nodes children of the same parent
- **External**: a node with no children also called a **leaf**
- **Internal**: a node with at least one child



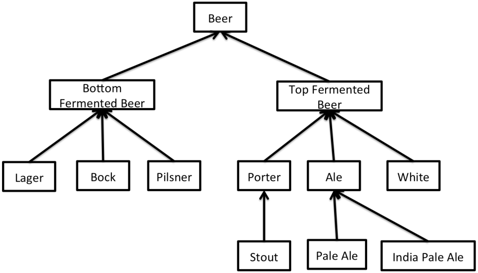
Other Node Relationships

- **Ancestor**: a node `u` is an ancestor of node `v` if `u=v` or `u` is an ancestor of the parent of `v`
- **Descendant**: a node `v` is a descendant of a node `u` if `u` is an ancestor of `v`
- **Subtree**: the subtree of a tree `T` rooted at a node `v` is the tree consisting of all the descendants of `v` in `T` (including `v`)

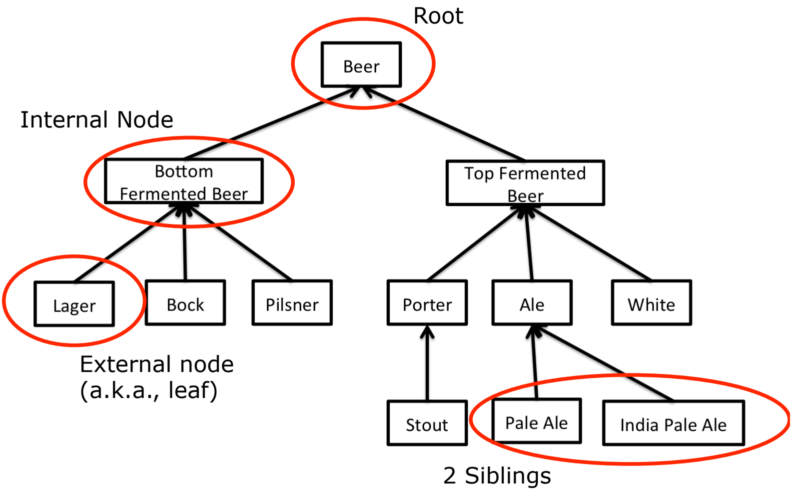


Edges and Paths in Trees

- **Edge**: an edge of tree `T` is a pair of nodes (`u,v`) such that `u` is the parent of `v`, or vice versa
- **Path**: a path of `T` is a sequence of nodes such that any two consecutive nodes in the sequence form an edge. There is only one path between any node and the root

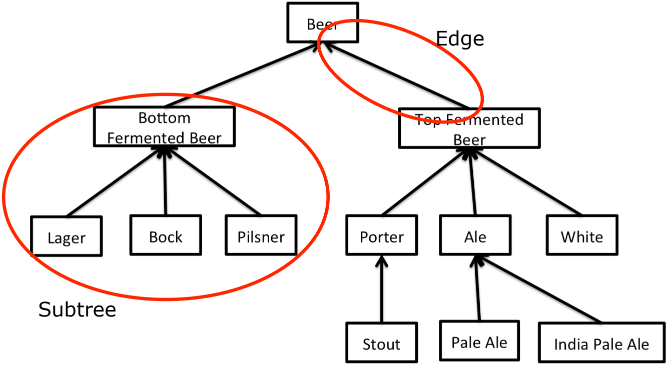


Example



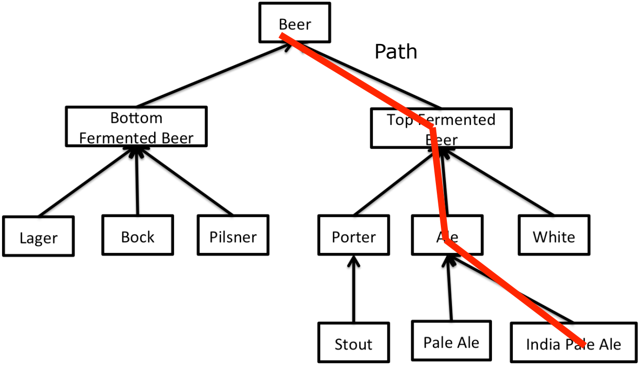
Root
Beer
Internal Node
Bottom Fermented Beer
Top Fermented Beer
Lager
Bock
Pilsner
Porter
Ale
White
External node
(a.k.a., leaf)
Stout
Pale Ale
India Pale Ale
2 Siblings

Example


Beer
Edge
Bottom Fermented Beer
Top Fermented Beer
Lager
Bock
Pilsner
Porter
Ale
White
Stout
Pale Ale
India Pale Ale
Subtree

12

Example


Beer
Path
Bottom Fermented Beer
Top Fermented Beer
Lager
Bock
Pilsner
Porter
Ale
White
Stout
Pale Ale
India Pale Ale

13

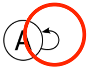
(not a) tree

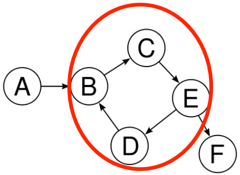
What looks like a tree but is **not** a tree?

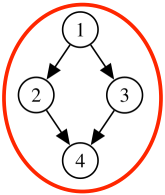
Cycles

Trees do not include directed or undirected **cycles**.

These are **not** trees:







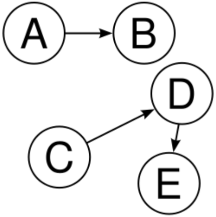
(not a) tree

What looks like a tree but is **not** a tree?

Non-connected components

Trees do not have non-connected parts.

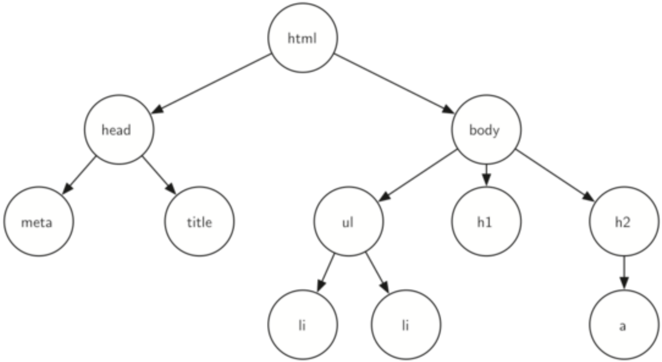
Not a tree (but sometimes referred to as a **forest**, i.e. a group of trees!)



Ordered Trees

A tree is ordered

if there is a meaningful linear order among the children of each node: the first, second, third etc. node. Such an order is usually visualised by arranging siblings left to right, according to their order.



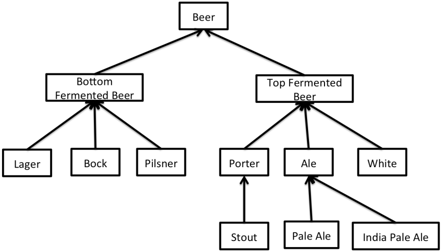
Depth and Height

Depth

How far away a node is from the root. The root of a tree has depth 0.

Height

The depth of the lowest leaf/external node. In other words, the length of the longest path from the root to a leaf. An empty tree has height 0.



Computing Height

Recursive function that adds one to height for each child along a path

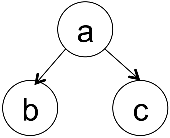
```
Algorithm height:
Input:  my_tree a tree
Output:  number of hops on longest path to a leaf
if my_tree is empty then
    return 0
else
    max_height ← 0
    for each child c of my_tree do
        max_height ← max (height(c), max_height)
    endfor
    return 1 + max_height
endif
```

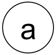
The Tree ADT

- `create_empty_tree()`: creates an empty tree
- `create_tree(n)`: creates a one node tree whose root is Node n
- `add_child(tree)`: add a subtree to the current tree
- `is_empty()`: determines whether a tree is empty
- `get_root()`: retrieves the Node that forms the root of a tree
- `remove_child(tree)`: "detaches" a subtree from the current tree

Tree ADT Usage Examples

```
my_tree =
(create_tree('a').add_child(create_tree('b'))).ad
d_child(create_tree('c'))
```

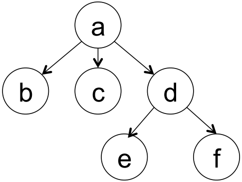


`my_tree.get_root()` $\Longrightarrow$ 

`my_tree.is_empty()` $\Longrightarrow$ `False`

Tree ADT Usage Examples

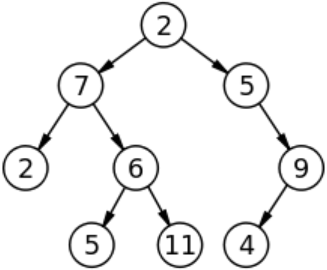
```
my_tree.  add_child((create_tree('d') \
.add_child(create_tree('e'))).add_child(create_tree('f')))
```

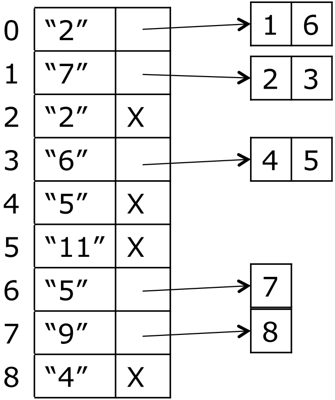


Array-based Representation of Trees

Array of arrays

Each representing the children of a node.



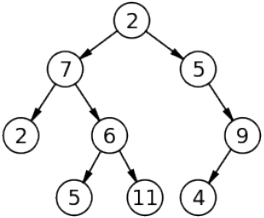


Array indices 0-8 are the nodes (top to bottom from left). Array contains node value and an array of child node indices. e.g. Root node has child nodes stored at indices 1 and 6.

Array-based Representation of Trees

A matrix

Can be very sparse if one node has lot of children.
Evolution likely to be easier to manage.



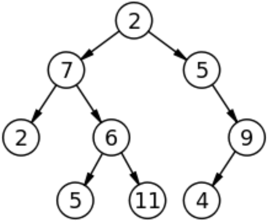
	label	0	1	2	3	4	5	6	7	8
0	"2"		1					1		
1	"7"			1	1					
2	"2"									
3	"6"					1	1			
4	"5"									
5	"11"									
6	"5"								1	
7	"9"									1
8	"4"									

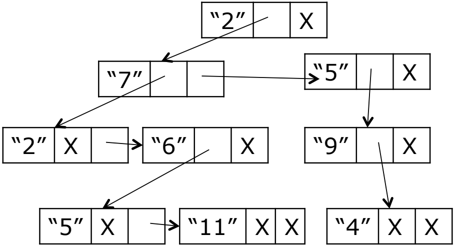
Linked List-based Representation of Trees

List of list of nodes

Always the same size as number of children

Using a doubly linked list



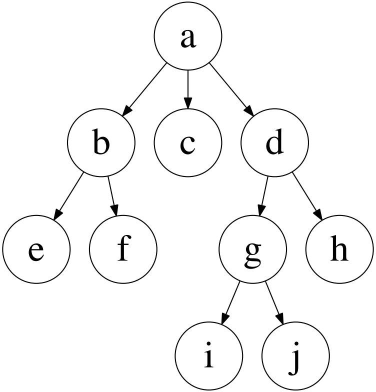


Searching a tree

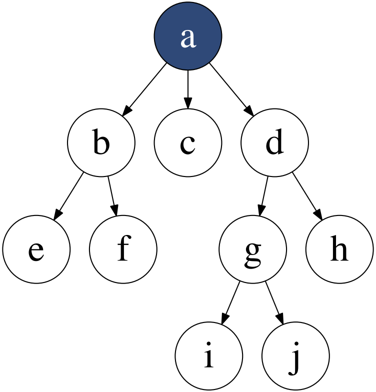
How to find the node you are looking for?

Go long – Depth First Search (DFS)
Go wide – Breadth First Search (BFS)

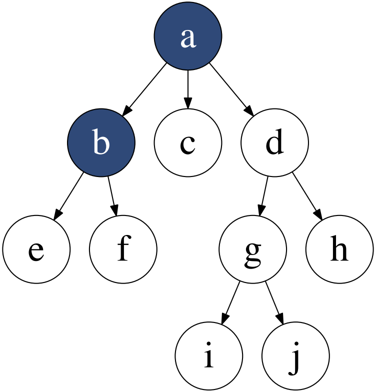
Depth First Search



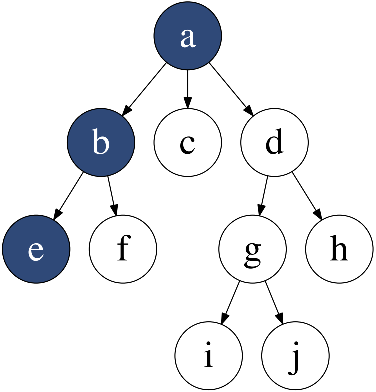
Depth First Search



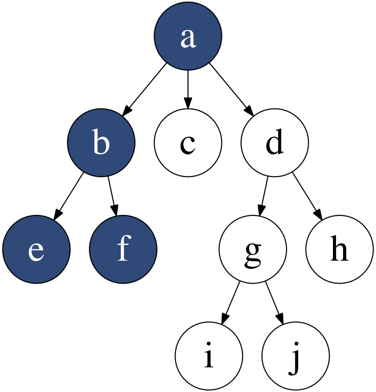
Depth First Search



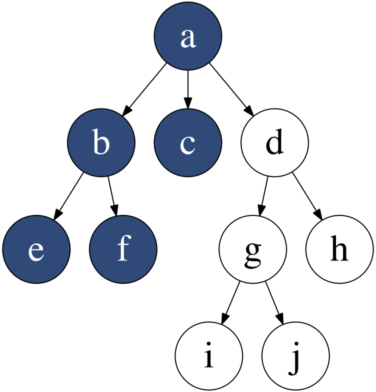
Depth First Search



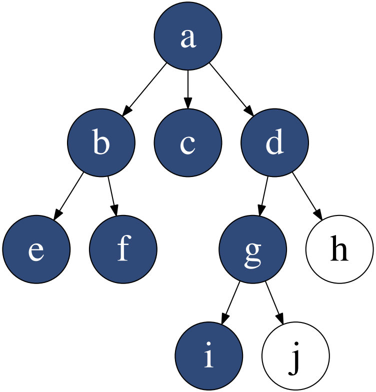
Depth First Search



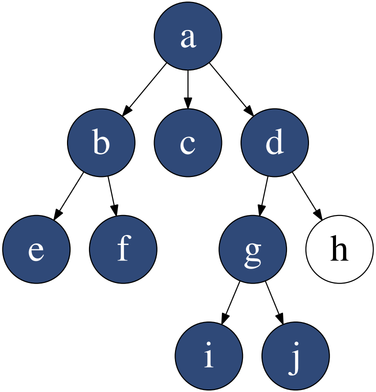
Depth First Search



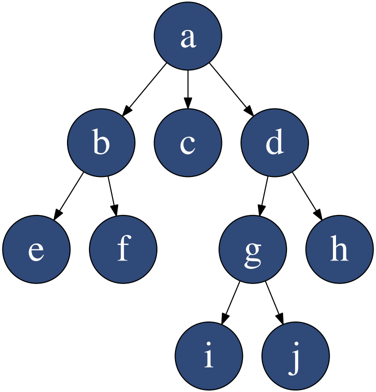
Depth First Search



Depth First Search



Depth First Search



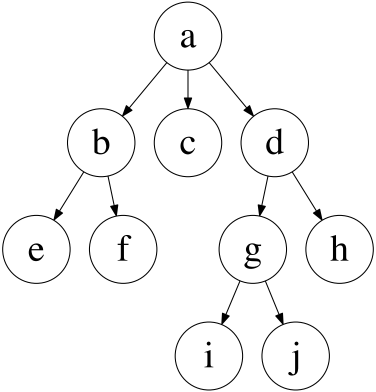
Depth First Search (recursive)

```
Algorithm dfs:
Input:  Tree t and node n
Output:  the function explores every node from n
if n is a leaf then # base case
    do something
else
for each child n_c of n do dfs(n_c)
        do something
    endfor
endif
```

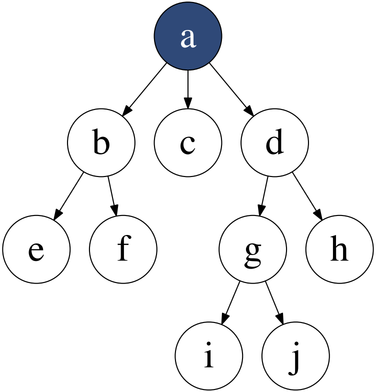
Depth First Search (non-recursive)

```
Algorithm dfs:
Input:  Tree t and node n
Output:  the function explores every node from n
to_visit ← empty stack
add n to to_visit
while to_visit is not empty do
    current ← pop to_visit # get the first element
    push all children of current to to_visit
    do something on current
endfor
```

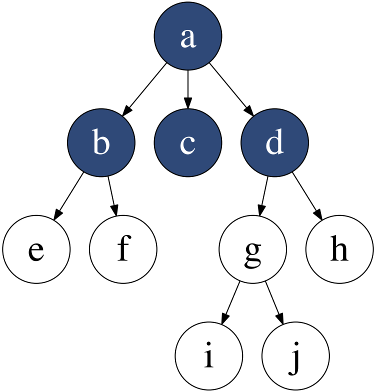
Breadth First Search



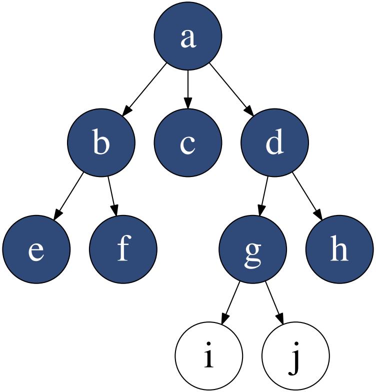
Breadth First Search



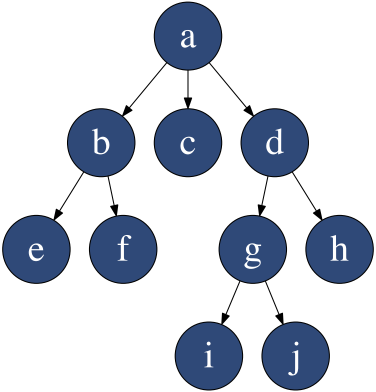
Breadth First Search



Breadth First Search



Breadth First Search



Breadth First Search (non-recursive)

```
Algorithm bfs:
Input:  Tree t and node n (root or not)
Output:  explores every node of t rooted at n
to_visit is a queue
enqueue n
while to_visit is not empty do
    n_current ← dequeue to_visit
    for each child n_c of n_current do
        enqueue n_c to to_visit
    endfor
    do something on n_current
endwhile
```

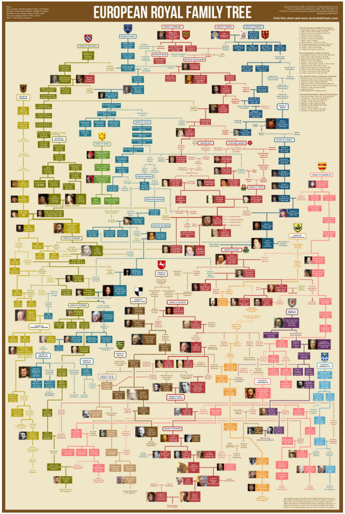
Breadth First Search (recursive)

```
Algorithm bfs:
Input:  queue q (originally having the root of the
tree)
Output:  explores every node of t rooted at n
if q is empty then # base case
    do something (?)
else
    current ← dequeue q
    for each child n_c of n do
        enqueue n_c
    endfor
    do something
    bfs(q)
endif
```

Conclusions

Tree vocabulary

non-linear structure, parent, child, sibling, forest, cycle, path, edge, subtree, internal node, external node, leaf, root node, ancestor, descendant, ordered, DFS, BFS



Trees and graphs are similar – both are non-linear data structures