

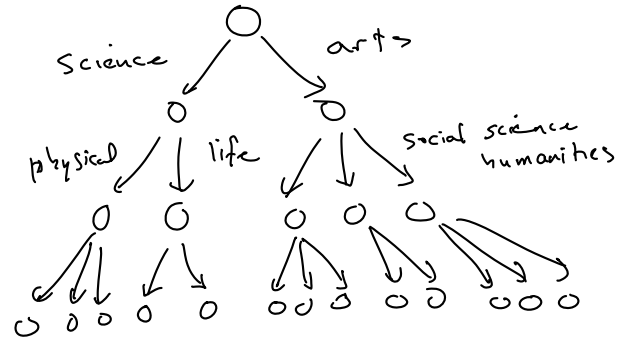
lecture 18

tree traversal

- depth first
(pre- vs post order)

- breadth first

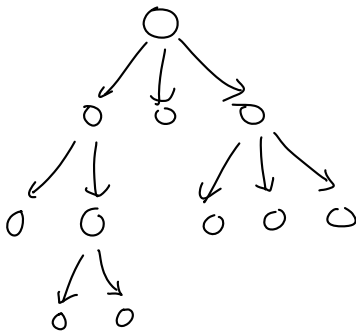
"Tree of Knowledge"



depth first
(know alot about
one subject)

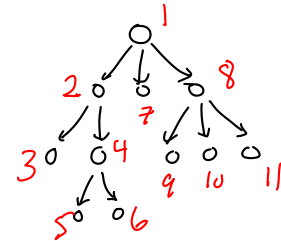
vs. breadth first
(know a bit about
many subjects)

Traversal - how to visit all
the nodes of a
tree?



pre-order (depth first) traversal

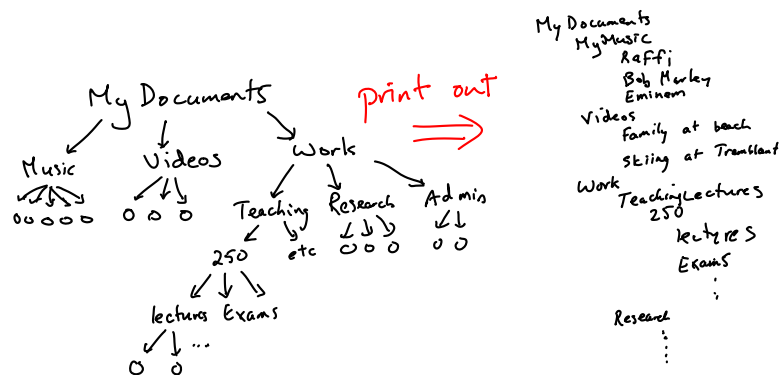
```
DFT (root) {
    visit root
    for each child of root
        DFT (root.child)
}
```



"visit a node" implies
you do something there.

Analogy, you aren't visiting
London if you are
just flying through
Heathrow airport.

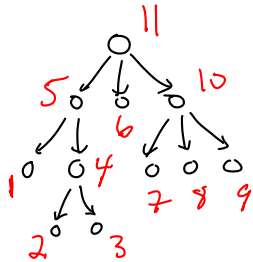
Example: Print file hierarchy



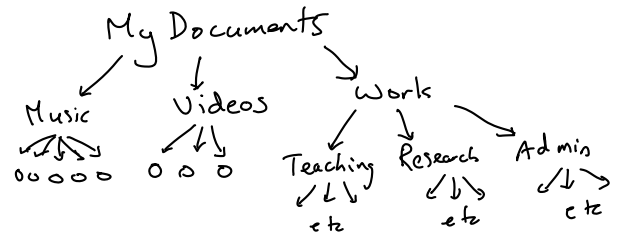
"post-order" (depth first) traversal

```

DFT (root) {
  for each child of root
    DFT (root.child)
  visit root
}
    
```



Example: Post order



What is the number of bytes in all files under directory My Documents?

numBytes (root) {

```

if root is a leaf
  return number of bytes at root
else {
  int sum = 0 // bytes
  for each child of root
    sum += numBytes (child)
  return sum
}
}
    
```

tree traversal

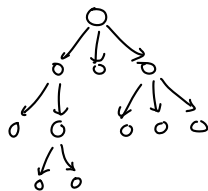
- recursive
 - depth first (pre- vs post order)
- non-recursive
 - depth first
 - breadth first

depth first traversal without recursion

DFT (root) {

```

S ← empty stack
S.push (root)
while (S is not empty) {
  cur ← S.pop()
  visit cur
  for each child of cur
    S.push (child)
}
}
    
```

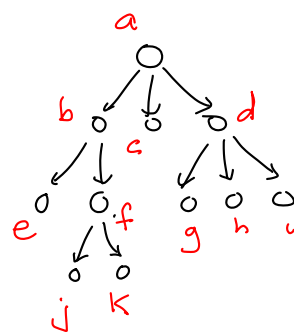


DFT (root) {

```

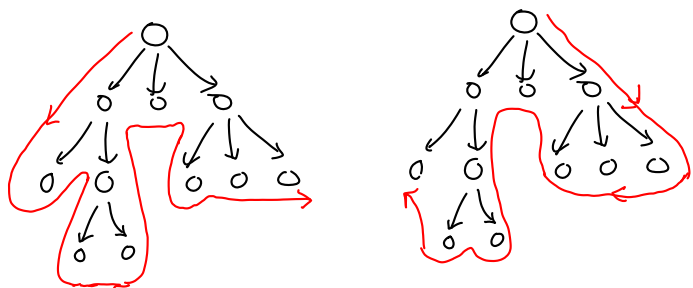
S ← empty stack
S.push (root)
while (S is not empty) {
  cur ← S.pop()
  visit cur
  for each child of cur
    S.push (child)
}
}
    
```

stack states entering while loop

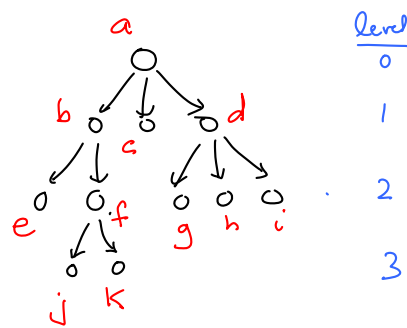


- a
- b c d
- b c g h i
- b c g h
- b c g
- b c
- b
- e f
- e j k
- e j
- e
-

- Non-recursive method above gives pre-order traversal but in "opposite" order to recursive method presented earlier



Breadth-first traversal



```
for each level i {
  visit all nodes at level i
}
```

Breadth-first traversal

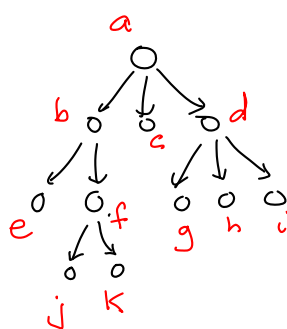
BFT (root) {

```
q ← empty queue
q.enqueue (root)
while (q is not empty) {
  cur ← q.dequeue()
  visit cur
  for each child of cur
    q.enqueue (child)
}
```

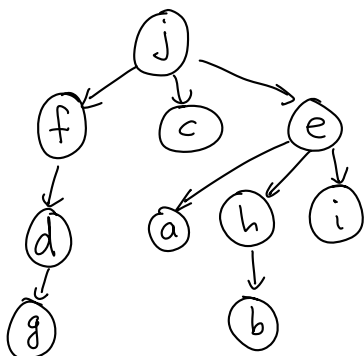
```
q ← empty queue
q.enqueue (root)
while (q is not empty) {
  cur ← q.dequeue()
  visit cur
  for each child of cur
    q.enqueue (child)
}
```

queue starts
entering while loop

a
bcd
cdef
def
efghi
fghi
ghijk
hijk
ijk
j
k
-



Example

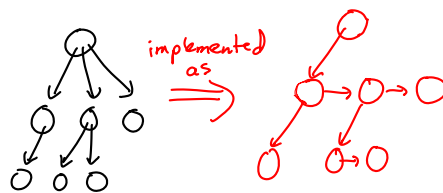


Depth first (recursive)

preorder: jfdgeeahbi
postorder: gdfcabhie j

Breadth first: jfcedahigb

Reminder - implementation (from last class)



"for each child" means:

```
tmp ← firstChild
while (tmp.nextSibling != null) {
  ...
}
```

```
class TreeNode <E> {
  E element
  TreeNode <E> firstChild
  TreeNode <E> nextSibling
  ...
}
```