Basic tree algorithms

- Depth

```
if T.isRoot(v) return 0;
   else return 1+depth(T, T.parent(v))
```

- Height

```
h=0;
for each v∈T.positions() do
   if T.isLeaf(v) h = max(h,depth(T,v));
   return h

if T.isLeaf(v) return 0
   else { h=0;
      for each w∈T.children(v) do
         h = max(h,height(T,w));
      return 1+h}
```

Basic tree algorithms

- Preorder traversal

```
visit (do something at) root
for each w∈T.children(v) do
   preorderTraverse(subtree T_w);
```

- Postorder traversal

```
for each w∈T.children(v) do
   postorderTraverse(subtree T_w);
visit (do something at) root
```

Example of preorder traversal

- Parenthetic representation of a tree
  If T consists of single node then $P(T) = \texttt{root}$
  Otherwise
  $P(T) = \texttt{root}(T) +$ "(" $+ P(T_1) + P(T_2) + \ldots + P(T_k) +$ ")"
- Can be produced by preorder traversal.

```
print root of T
if isInternal(root) {
   print "(";
   for each children w of root
      call parentheticRep(T_w);
   print ")";
}
```

Example of postorder traversal

- Computing size of individual files in a file system.

```
int s=size(root)
if isExternal(root) {
   print s;
   return s;
}
for each children w of root {
   s+computeFileSize(T_w);
}
print s;
return s;
```

Evaluation of tree expression

```
if (isInternal(v)) {
   x=evalExpression(leftSubTree(T_v))
   y=evalExpression(rightSubTree(T_v))
   // assume content(v) = op (operator)
   return x op y
}
else
   return val(v)
```

Inorder traversal

for subtrees $T_1, T_2, \ldots T_k$ of T do{
   inorderTraverse(subtree T_i);
   visit root r of T
}

- Means that `root` is visited in between traversal of each subtree.
- In case of preorder traversal root is visited before visiting any subtree, whereas in case of postorder traversal root is visited last.
- Euler tour simplifies all traversals.