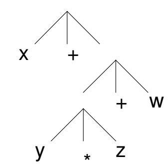
CSE 326
Lecture 6

Spring Quarter 1997

Trees

- Various kinds of trees are basic data types in many, many applications.
- Parse Tree for x +y*z + w



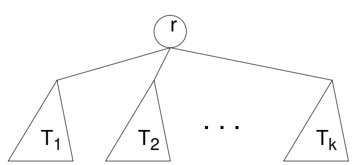
- Organizational Trees - Unix file system
- Search Trees - Better than a linked list for "insert", "delete", "find.
- Set Union Find Trees - Good for keeping sets of disjoint sets efficiently

Tree Terminology

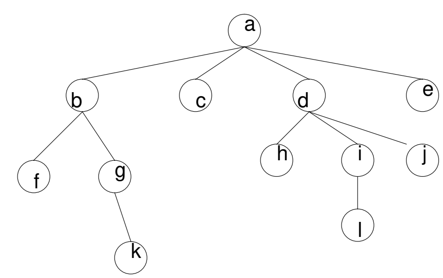
- Recursive Definiton of trees

  - Empty tree Λ has no root.

  - Given trees T_1, T_2, ..., T_k and a node r a treeT is formed by making the root r whose children are the roots of T_1, T_2, ... , T_k respectively and the parent of the roots of T_1, T_2, ... , T_k is r.

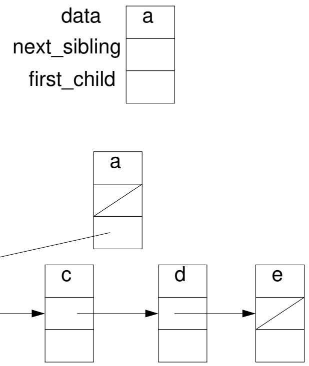


Example

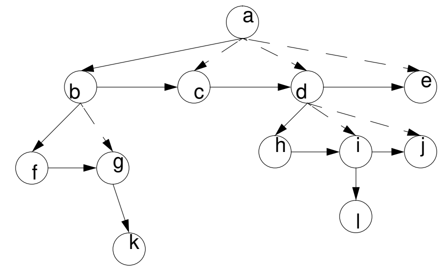


- root
- node
- children
- parent
- sibling
- descendent
- ancestor
- subtree
- leaf
- depth
- height

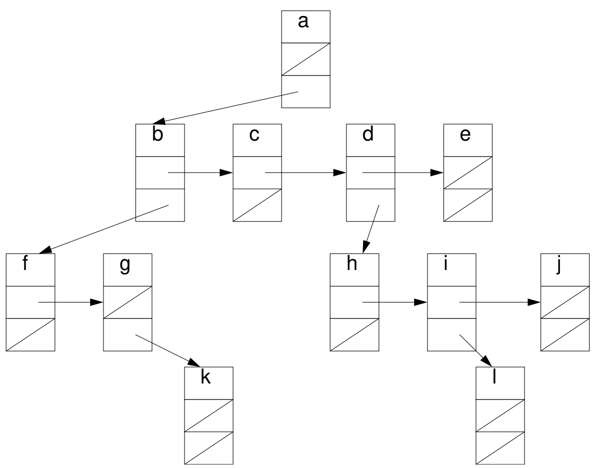
Basic Tree Data Structure



View of Tree Data Structure



Actual Data Structure for the Tree

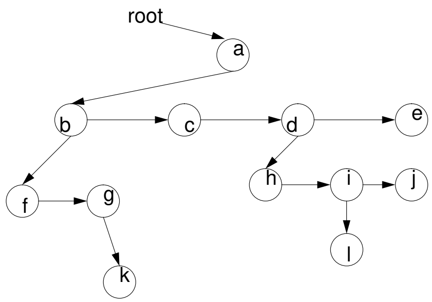


Preorder Traversal

- Preorder - Visit the node, then recursively visit each of the subtrees.

```
preorder(n) {
local node pointer c;
if not(n=null) then {
   Do_something;
   c := n.first_child;
   while not (c=null) do {
     preorder(c);
     c := c.next_sibling}}}
```

Preorder Traversal Example



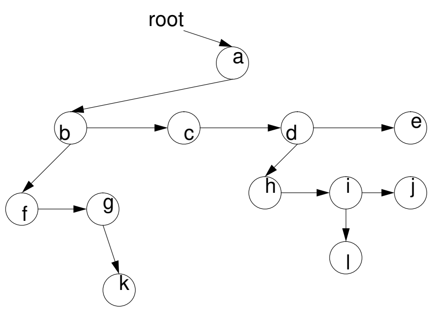
Computing the Depth of all Nodes

- Add a field called "depth"

```
Depth(n,d) {
local node pointer c;
if not(n=null) then {
  n.depth :=d;
  c := n.first_child;
  while not (c=null) do {
      Depth(c,d+1)
      c := c.next_sibling}}}
```

- Call Depth(root,0) to set the depth field correctly.

Depth Calculation

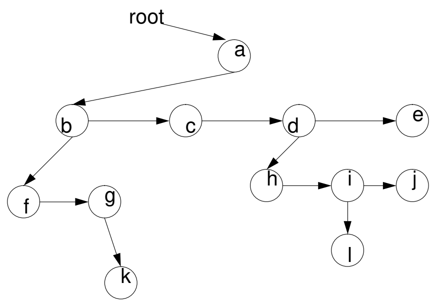


Postorder Traversal

- Postorder - Recursively visit each of the subtrees, then visit the node.

```
postorder(n) {
local node pointer c;
if not(n = null) then {
  c := n.first_child;
  while not(c=null) do {
     postorder(c);
     c := c.next_sibling};
  Do_something}}
```

Postorder Traversal Example



Computing the Size of a Tree with a Function

- count(n) {

  local node pointer c, integer m;

  ```
  if n = null then return(0) else {
      c := n.first_child;
      m := 0;
      while not (c=null) do {
          m := m + count(c);
          c := c.next_sibling}
      return(m+1) }}
  ```
- Call count(root) to compute the number of nodes in the tree.

Size of Tree Example