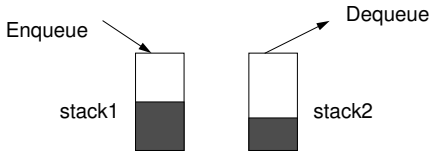
CSE 326
Lecture 9

Spring Quarter 1995

Amortized Time

- Instead of looking at the worst case time per operation in a data structure we look at the worst cast time fro n operations.
- Example: Implementing a queue using two stacks



- Enqueue - push into stack1
- Dequeue - remove from stack2 unless it is empty. If stack2 is empty then move all the contents of stack1 to stack2

Example

- Operations: E(1), E(2), E(3), D, D, E(4), E(5), D, D, E(6)
- Costs:

Amortized Analysis

- Starting with an empty queue, how many stack operations does it take to do n queue operation?

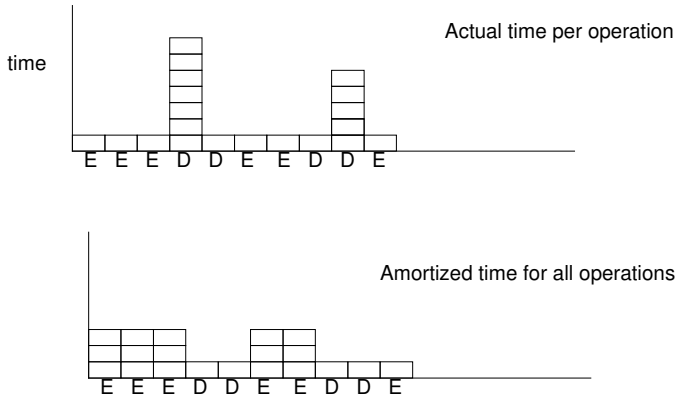
  Answer: O(n) even though some individual Dequeue operations take many stack operations.
- Why? Each element that enters the queue goes through at most four stack operations.

  push1 on entry to the queue

  pop1 and push2 to move it from stack1 to stack2

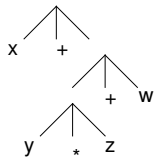
  pop2 to remove it from the queue
- Total number of stack operations for n queue operations is bounded by 3n

Picture of Amortization



Trees

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



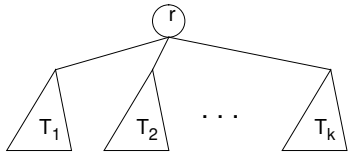
- Organizational Trees - Unix file system
- Search Trees - Better than a linked list for "insert", "delete", "find.
- Set Union Find Trees - Good for keeping sets of disjoint sets efficiently

Tree Terminology

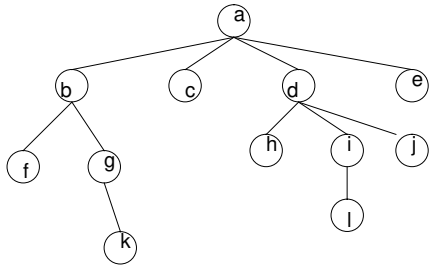
- Recursive Definiton of trees

  - 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.

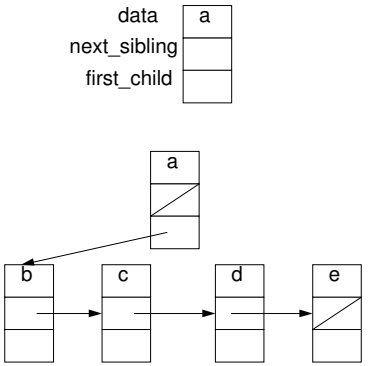


Example

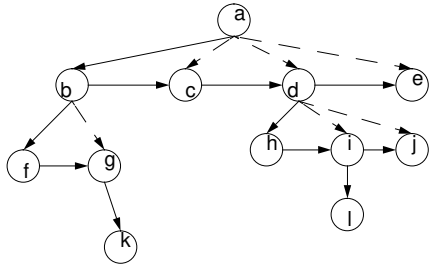


- root
- node
- children
- parent
- sibling
- descendent
- ancestor
- subtree
- leaf
- depth
- height

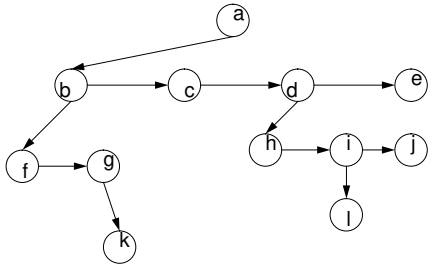
Basic Tree Data Structure



View of Tree Data Structure



Actual Data Structure for the Tree

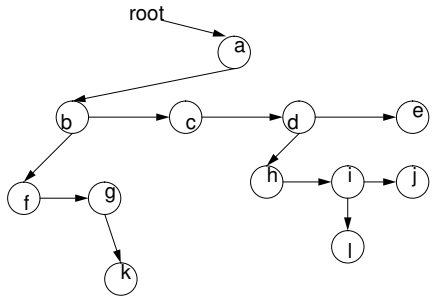


Preorder Traversal

- 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



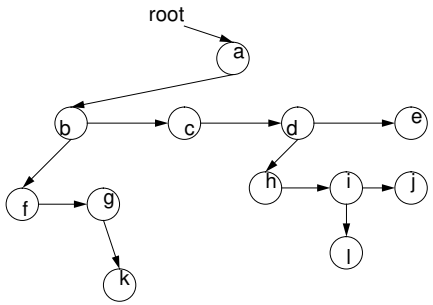
Computing the Depth of all Nodes

- 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

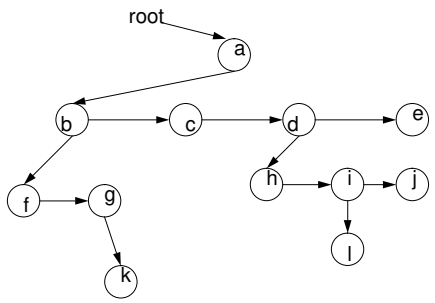


Postorder Traversal

- 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



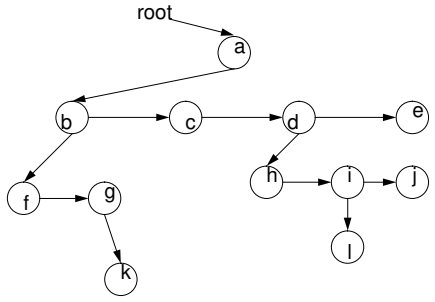
Computing the Height of Each Node

- We have a new field "height" for each node

```
Height(n) {
local node pointer c;
if not(n=null) then {
   h := 0
   c := n.first_child;
   while not (c=null) do {
      h := max(h,c.height);
      c := c.next_sibling};
    n.height := h }}
```

- Call Height(root) to correctly set all height fields

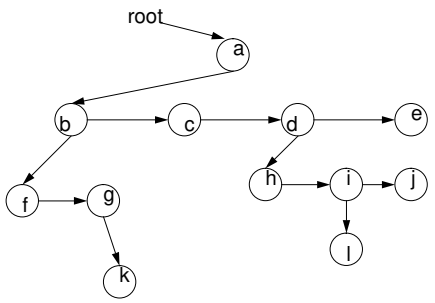
Height Example



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 := 1;
   while not (c=null) do {
      m := m + count(c);
      c := c.next_sibling}
   return(m) }}
```

- Call count(root) to compute the number of nodes in the tree.

Size of Tree Example