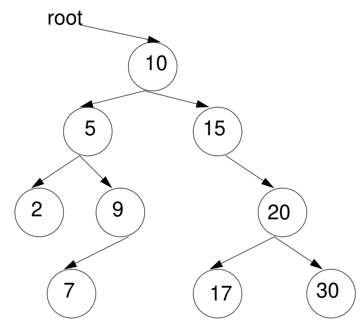
CSE 326
Lecture 12

Spring Quarter 1995

Examples of Finding Minimum and Maximum



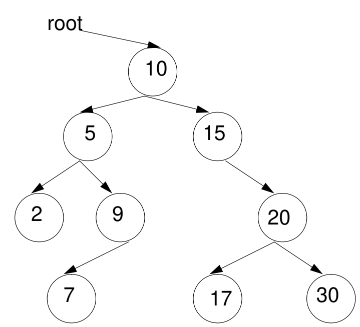
Minimum and Maximum

- Min(n) returns a pointer to the node in the subtree rooted at n with the smallest data. We assume that n points to a node.
- Min(n) {

  if n.left = null then return(n)
  else return Min(n.left) }
- To find the minimum in the entire tree call Min(root).
- Max(n) is similar but looking right instead of left.
- Min and Max exhibit end recursion so iterative programs are easy.
- Min(n) {

  while not(n.left = null) do n := n.left;
  return(n)}

Finding the Successor and Predecessor



Successor and Predecessor

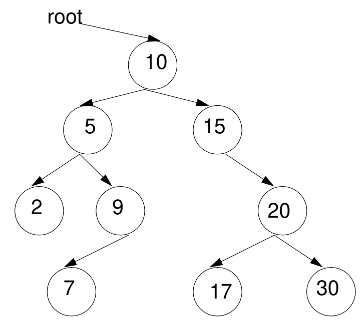
- Successor(n) finds the node whose data is next largest after n's data in the subtree rooted at n. Assume that n has a successor.
- Successor(n) {

  return(Min(n.right)) }
- Iterative version

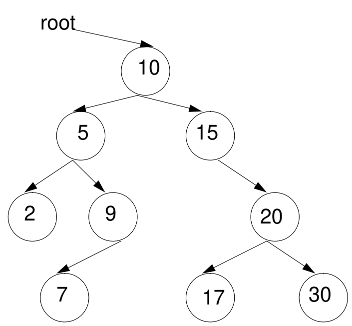
  Successor(n) {

  n := n.right;
  while not(n.left = null) do n := n.left;
  return(n) }
- For Predecessor just reverse the roles of left and right.

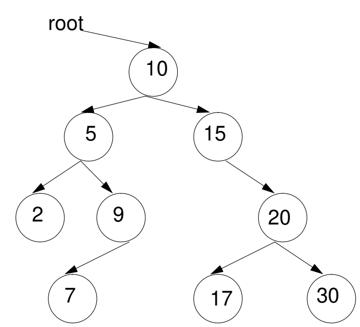
Deletion - Leaf Case



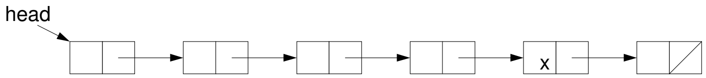
Deletion - One Child Case



Deletion - Two Child Case



Approaches to Deletion in a List



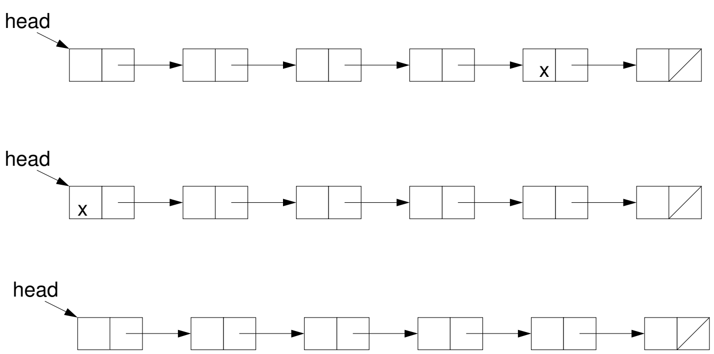
- Iterative
- Recursive, finding node before
- Recursive, using reference paramenters

Iterative

- Delete(x) removes x from the list pointed to by head.
- Delete(x) {

```
local pointer p;
if not(head=null) then {
  if p.data = x then head := head.next
  else {
    p := head;
    while not(p.next = null) do
      if not(p.next.data = x) then p := p.next;
    if not(p.next=null) then p.next := p.next.next}}}
```


Iterative Delete Example



Recursion Finding the Node Before

- Find_Before(x,p) returns the node before the node with x in it in the list pointed to by p or returns null if there is no such node. We assume that p points to a node.
- Find_Before(x,p) {

```
case
  p.next = null : return(null);
  p.next.data = x : return(p);
  otherwise : return(Find_Before(x, p.next))
endcase }
```

- Delete(x) {

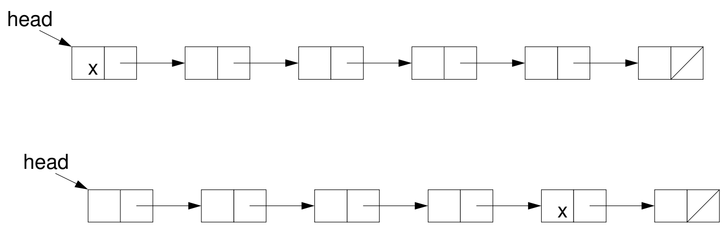
```
if not(head=null) then {
  if head.data = x then head := head.next
  else {
    p := Find_Before(x,head);
    if not(p.next = null) then p.next := p.next.next }
```

Recursion Using Reference Parameters

- A reference parameter is the actual value which is not copied.
- Delete(x, ref p) {

```
if not(p = null) then
  if p.data = x then p := p.next
  else Delete(x,p.next) }
```

- Call Delete(x,head) to delete x.



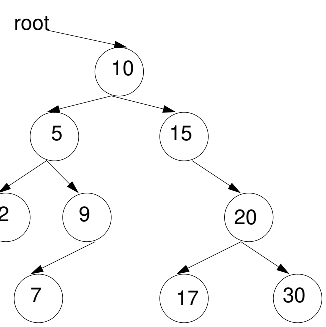
Back to Tree Deletion

- Recursive reference parameter technique
- Delete(x, ref p) {

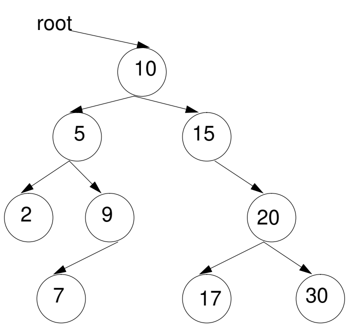
```
local pointer q;

if not(p=null) then
  case
    p.data > x: Delete(x, p.right_child);
    p.data < x: Delete(x. p.left_child);
    p.data = x:
        case
          p.left_child := null: p := p.right_child;
          p.right_child := null: p := p.left_child;
          otherwise: {
              q := Successor(p);
              p.data := q.data;
              Delete(q.data,p.right_child)}
         endcase
   endcase }
```


Example of Deletion



Another Example of Deletion



Value vs. Reference Parameters

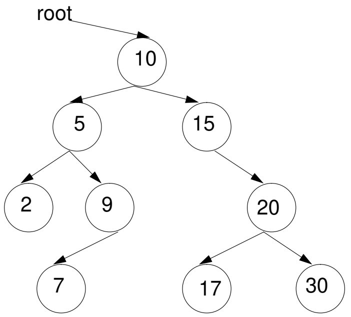
- Value parameters are generally preferred because there are no side effects.
- When to use Reference parameters

  1. The parameter is an array or some large object that needs to be acted on directly.

  2. The parameter is a pointer that will need to be changed as a result of the call. Because unintentioned side effects care must be taken!

Iterative Delete by Example

- Three pointers are used: "leader", "parent", and "waiter". It is tricky code with much potential for problems.



General Case of Iterative Delete