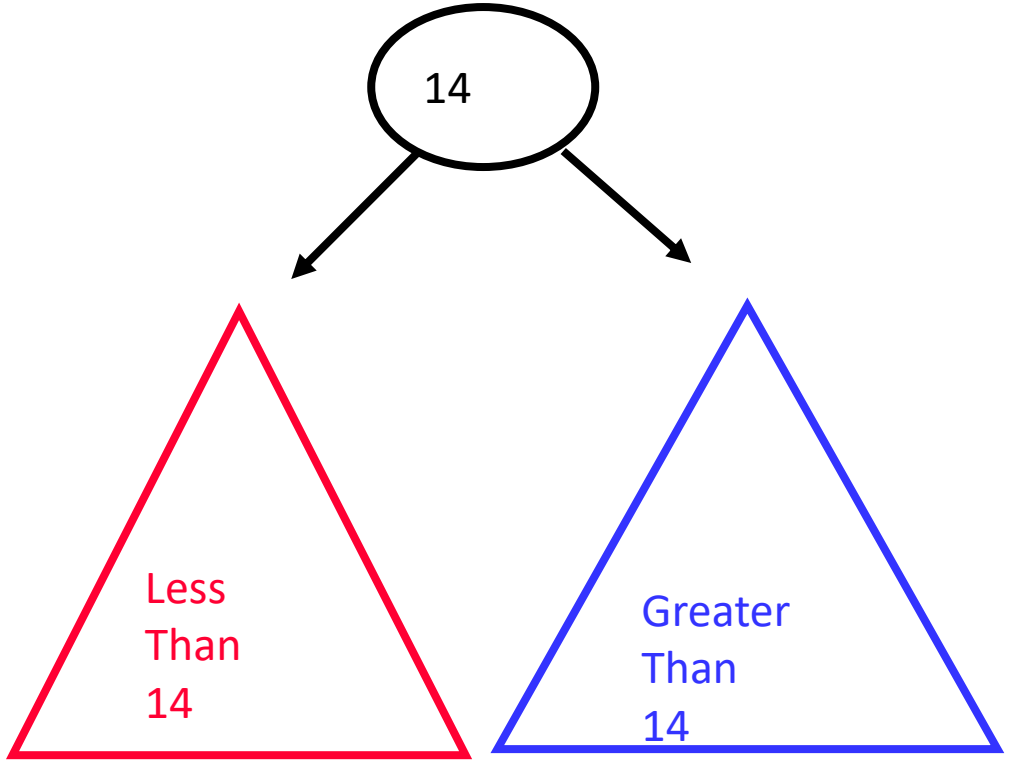
SEARCH A NODE

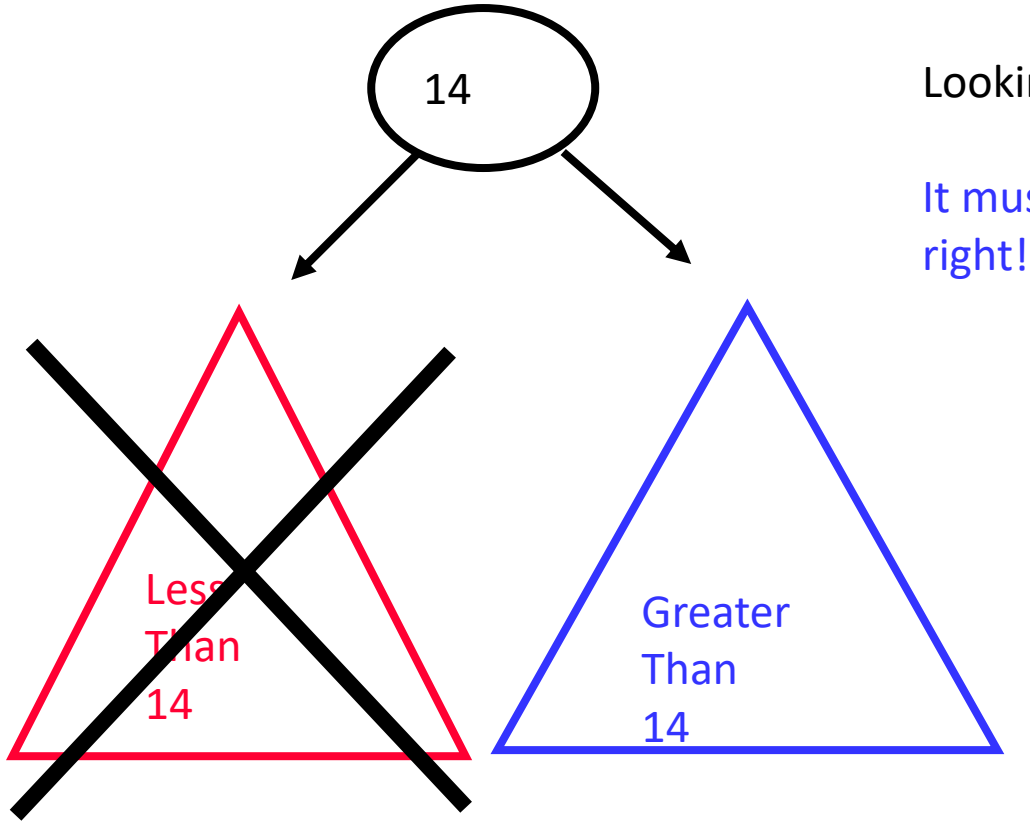
Search a Node

- **We've got a Binary Search Tree and we want to determine if an element is in the collection.**



Cutting the Work in Half

- **In searching for a match, we can ignore half of the tree at each comparison.**



Looking for 42…

It must be to the right!

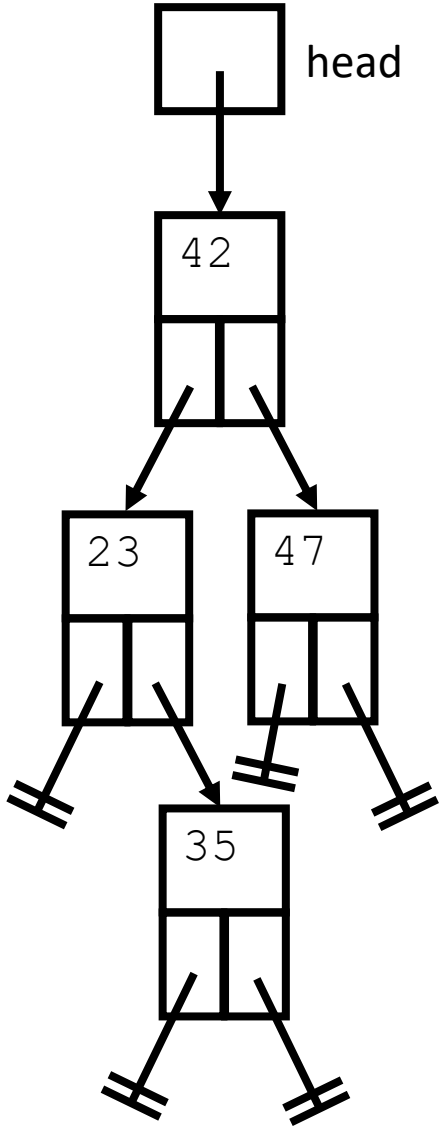
The Binary Search Algorithm

```
if at NIL // not found
  DO NOT FOUND WORK
elseif ( match ) then  // found
  DO FOUND WORK
elseif ( value to match < current value )
  recurse left   // must be to left
else
  recurse right  // must be to right
```

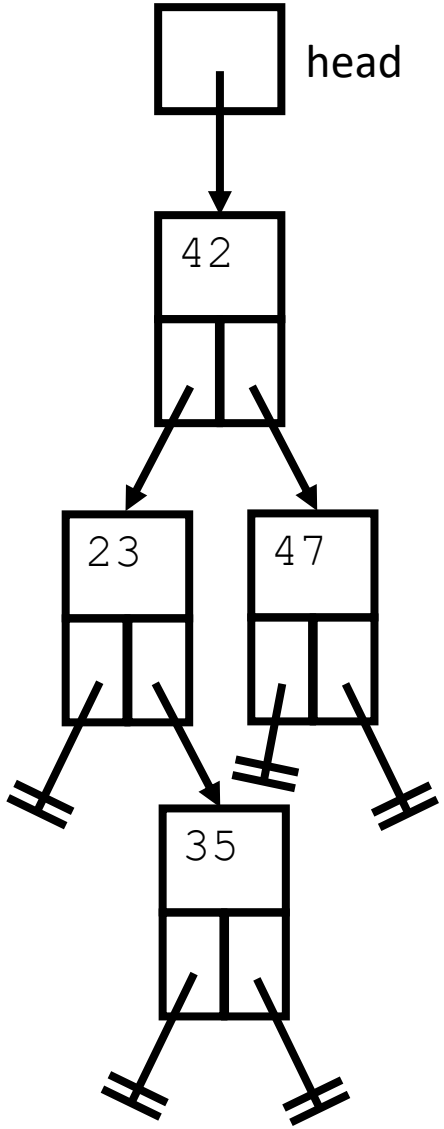
The Binary Search for a BST

```
procedure Search(cur iot in Ptr toa
 Node,
                 target isoftype in Num)
  if(cur = NIL) then
    print("Not Found")
  elseif(cur^.data = target)
    print("Target Found")
  elseif(cur^.data > target)
    Search(cur^.left, target)
  else
    Search(cur^.right, target)
  endif
endprocedure // Search
```

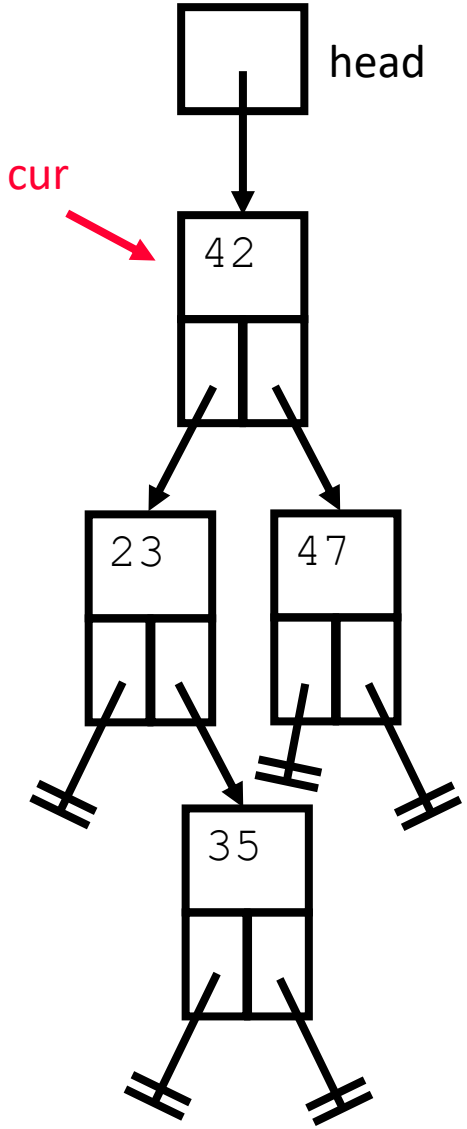
```
.
.
Search(head, 35)
Search(head, 87)
.
.
```



```
.
.
Search(head, 35)
Search(head, 87)
.
.
```

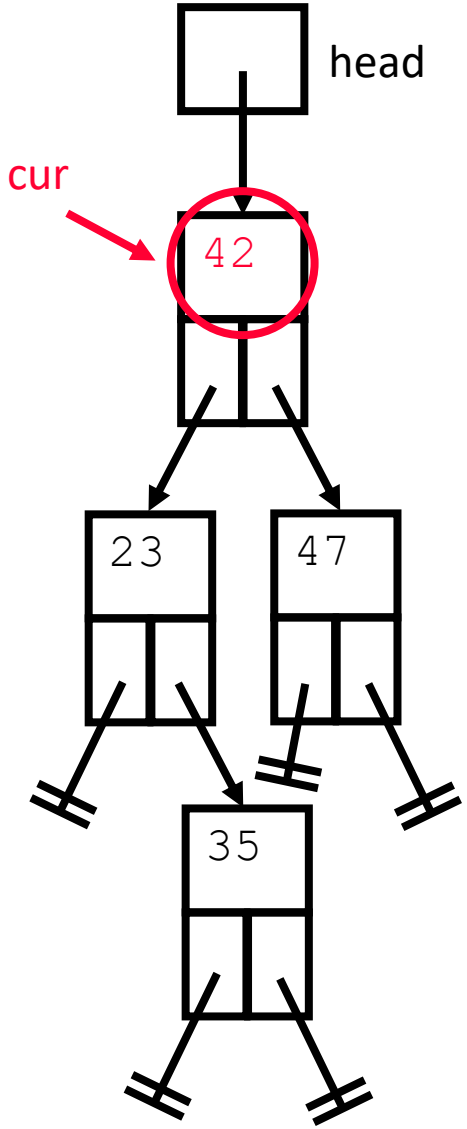


```
Procedure Search(
 cur iot in Ptr toa Node,
 target iot in num)
 if(cur = NIL) then
  print("Not Found")
 elseif(cur^.data = target)
  print("Target Found")
 elseif(cur^.data > target)
  Search(cur^.left, target)
 else
  Search(cur^.right, target)
 endif
endprocedure // Search
```



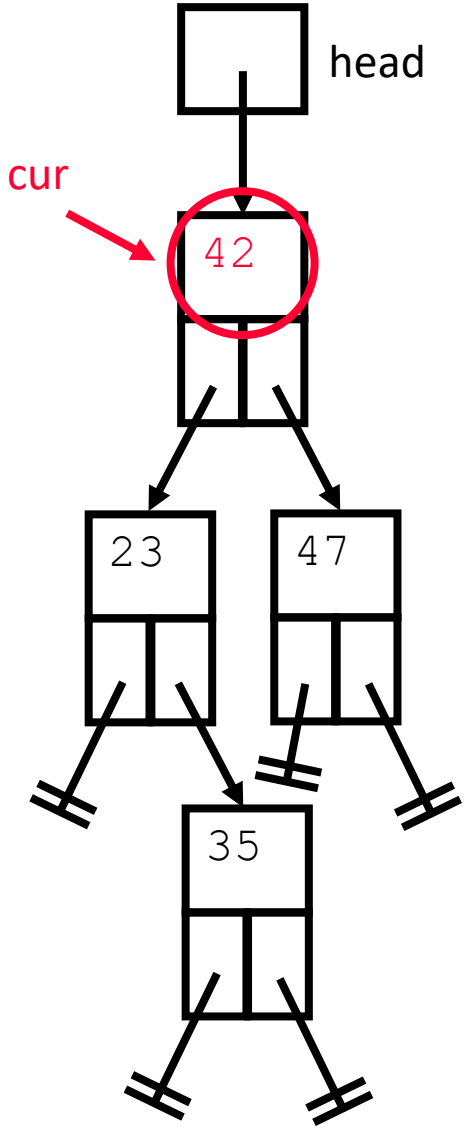
target = 35

```
Procedure Search(
 cur iot in Ptr toa Node,
 target iot in num)
 if(cur = NIL) then
  print("Not Found")
 elseif(cur^.data = target)
  print("Target Found")
 elseif(cur^.data > target)
  Search(cur^.left, target)
 else
  Search(cur^.right, target)
 endif
endprocedure // Search
```



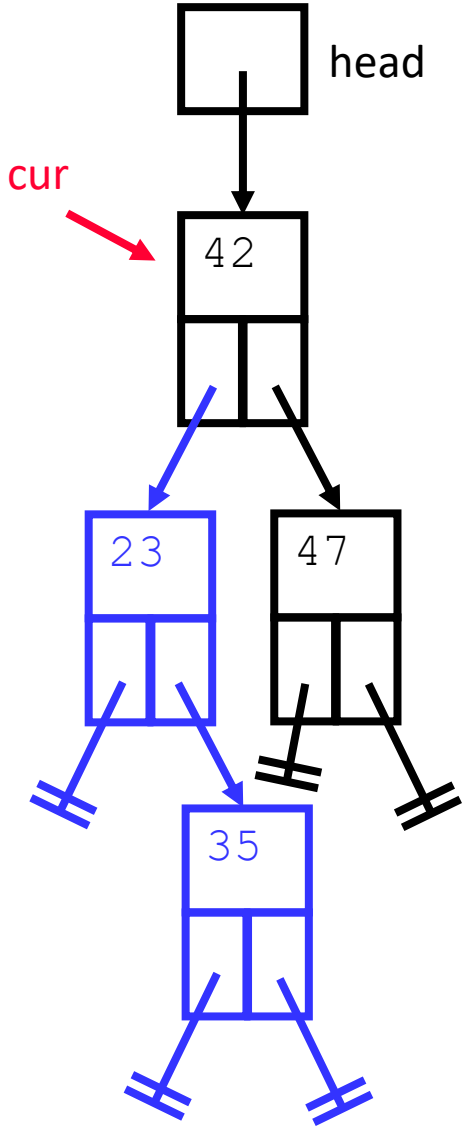
target = 35

```
Procedure Search(
 cur iot in Ptr toa Node,
 target iot in num)
 if(cur = NIL) then
  print("Not Found")
 elseif(cur^.data = target)
  print("Target Found")
 elseif(cur^.data > target)
  Search(cur^.left, target)
 else
  Search(cur^.right, target)
 endif
endprocedure // Search
```



target = 35

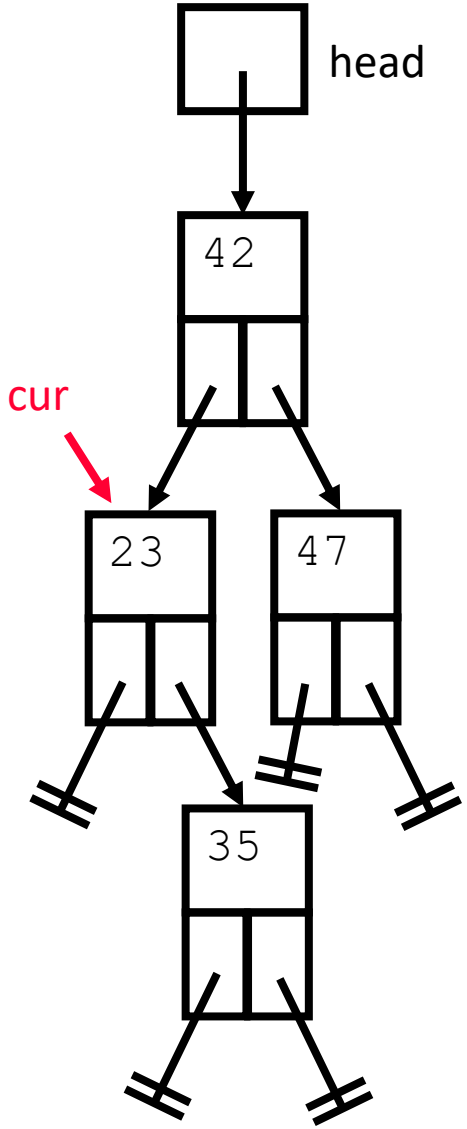
```
Procedure Search(
 cur iot in Ptr toa Node,
 target iot in num)
 if(cur = NIL) then
  print("Not Found")
 elseif(cur^.data = target)
  print("Target Found")
 elseif(cur^.data > target)
  Search(cur^.left, target)
 else
  Search(cur^.right, target)
 endif
endprocedure // Search
```



target = 35

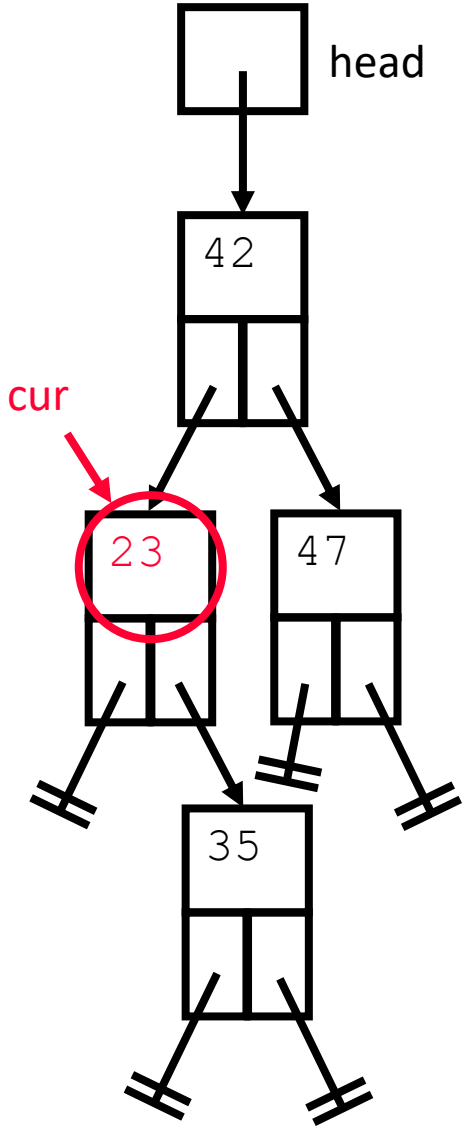
```
Procedure Search(
 cur iot in Ptr toa Node,
 target iot in num)
 if(cur = NIL) then
  print("Not Found")
 elseif(cur^.data = target)
  print("Target Found")
 elseif(cur^.data > target)
  Search(cur^.left, target)
 else
  Search(cur^.right, target)
 endif
endprocedure // Search
```

target = 35



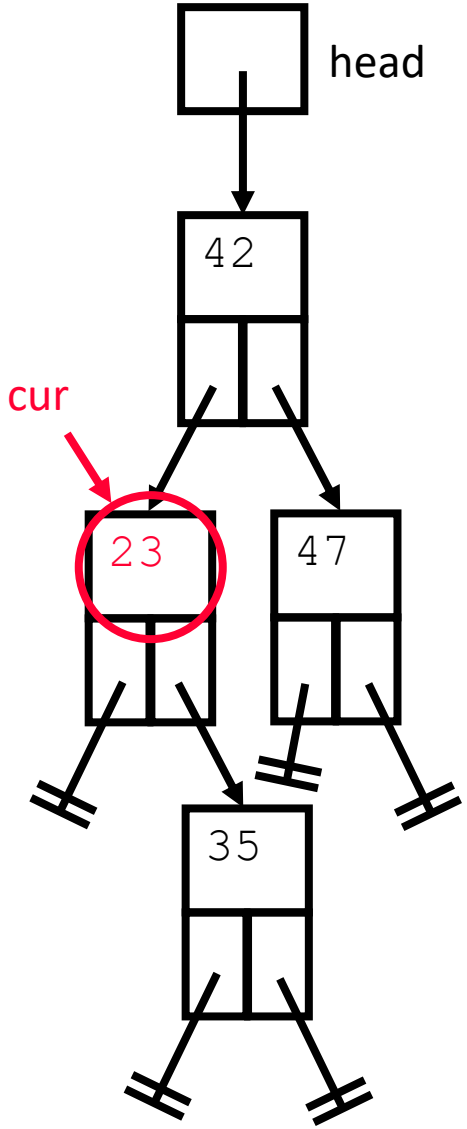
```
Procedure Search(
 cur iot in Ptr toa Node,
 target iot in num)
 if(cur = NIL) then
  print("Not Found")
 elseif(cur^.data = target)
  print("Target Found")
 elseif(cur^.data > target)
  Search(cur^.left, target)
 else
  Search(cur^.right, target)
 endif
endprocedure // Search
```

```
target = 35
```



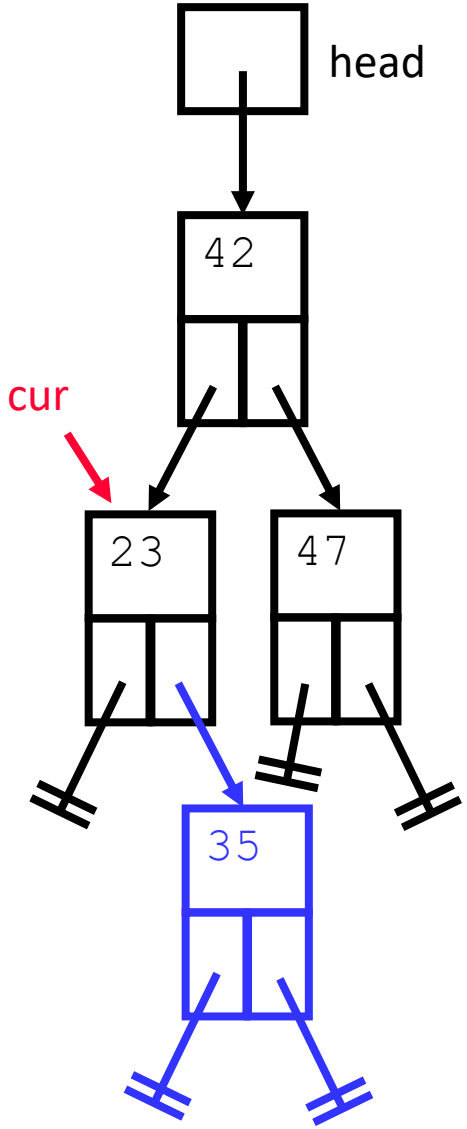
```
Procedure Search(
 cur iot in Ptr toa Node,
 target iot in num)
 if(cur = NIL) then
  print("Not Found")
 elseif(cur^.data = target)
  print("Target Found")
 elseif(cur^.data > target)
  Search(cur^.left, target)
 else
  Search(cur^.right, target)
 endif
endprocedure // Search
```

```
target = 35
```



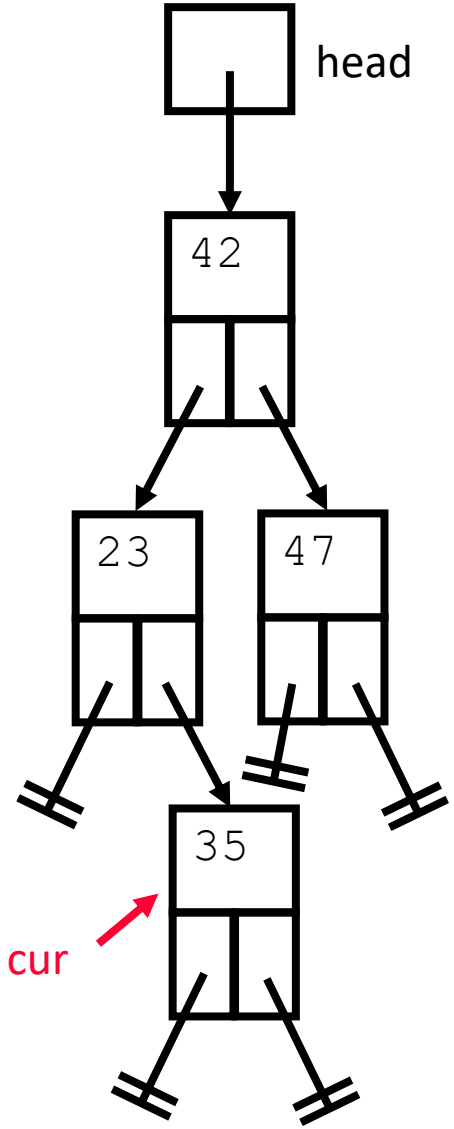
```
Procedure Search(
 cur iot in Ptr toa Node,
 target iot in num)
 if(cur = NIL) then
  print("Not Found")
 elseif(cur^.data = target)
  print("Target Found")
 elseif(cur^.data > target)
  Search(cur^.left, target)
 else
  Search(cur^.right, target)
 endif
endprocedure // Search
```

```
target = 35
```



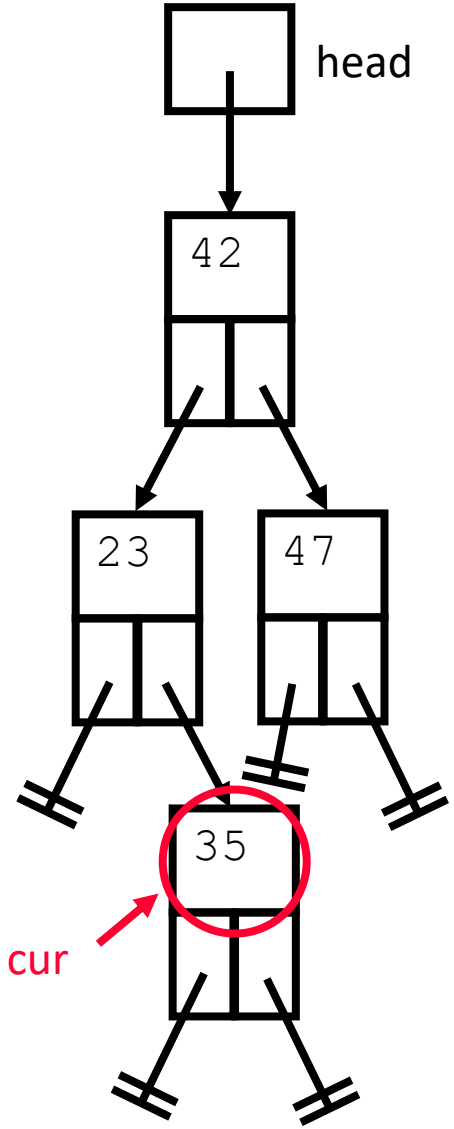
```
Procedure Search(
 cur iot in Ptr toa Node,
 target iot in num)
 if(cur = NIL) then
  print("Not Found")
 elseif(cur^.data = target)
  print("Target Found")
 elseif(cur^.data > target)
  Search(cur^.left, target)
 else
  Search(cur^.right, target)
 endif
endprocedure // Search
```

```
target = 35
```



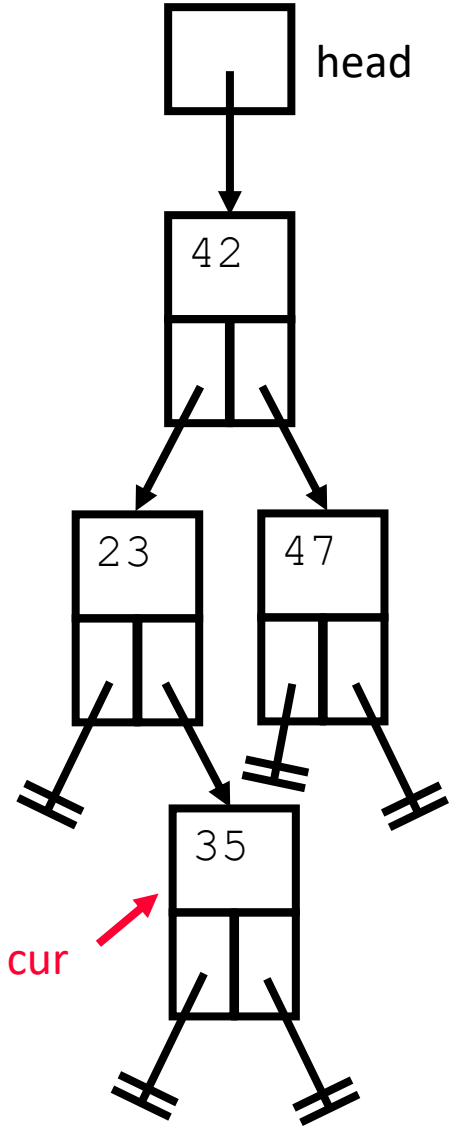
```
Procedure Search(
 cur iot in Ptr toa Node,
 target iot in num)
 if(cur = NIL) then
  print("Not Found")
 elseif(cur^.data = target)
  print("Target Found")
 elseif(cur^.data > target)
  Search(cur^.left, target)
 else
  Search(cur^.right, target)
 endif
endprocedure // Search
```

target = 35

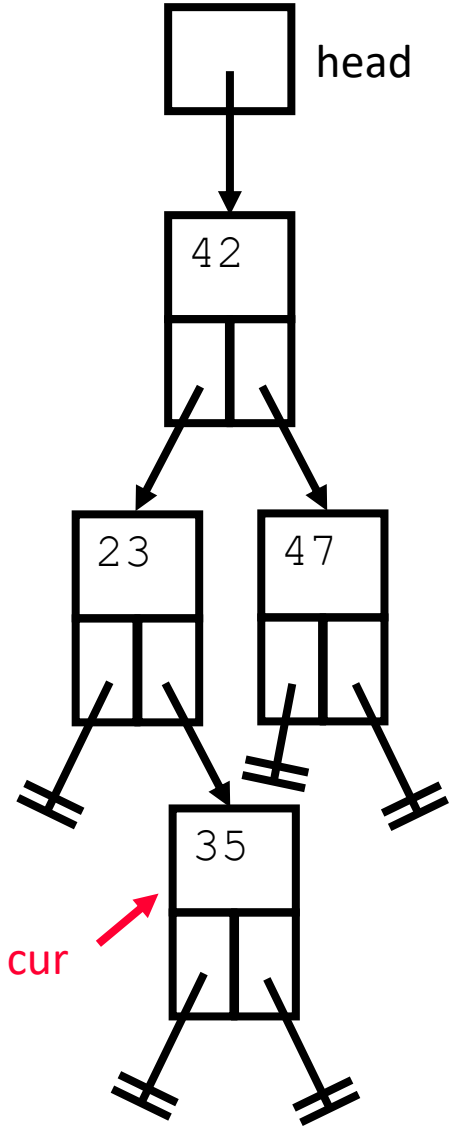


```
Procedure Search(
 cur iot in Ptr toa Node,
 target iot in num)
 if(cur = NIL) then
  print("Not Found")
 elseif(cur^.data = target)
  print("Target Found")
 elseif(cur^.data > target)
  Search(cur^.left, target)
 else
  Search(cur^.right, target)
 endif
endprocedure // Search
```

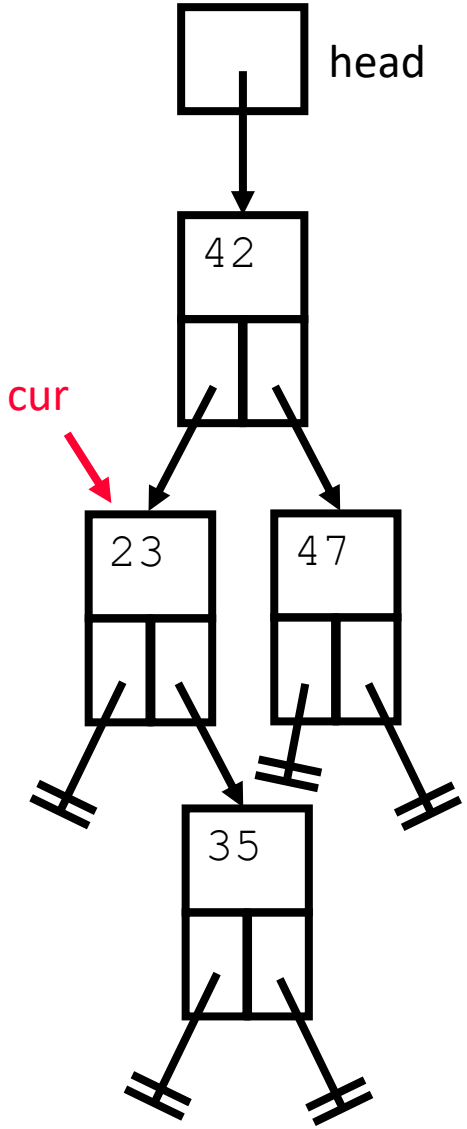
Target Found



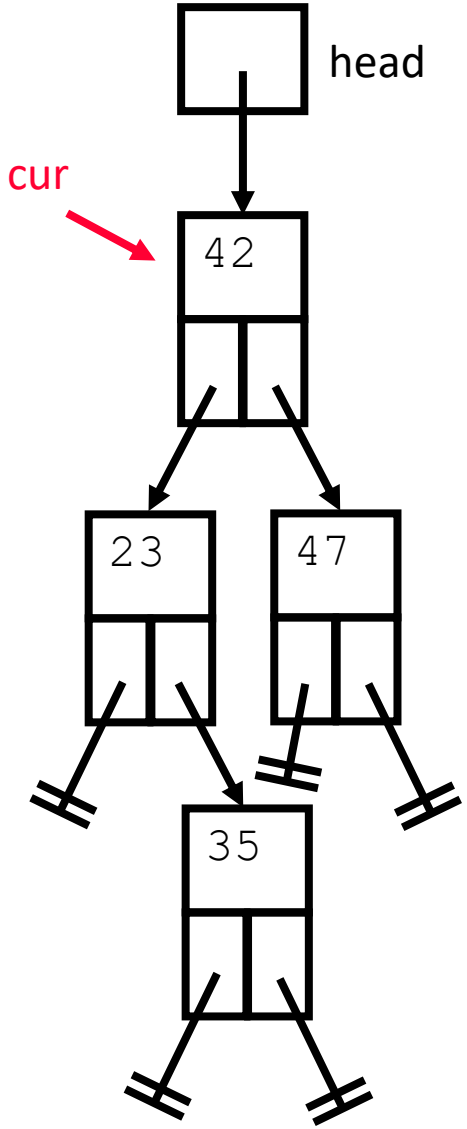
```
Procedure Search(
 cur iot in Ptr toa Node,
 target iot in num)
 if(cur = NIL) then
  print("Not Found")
 elseif(cur^.data = target)
  print("Target Found")
 elseif(cur^.data > target)
  Search(cur^.left, target)
 else
  Search(cur^.right, target)
 endif
endprocedure // Search
```



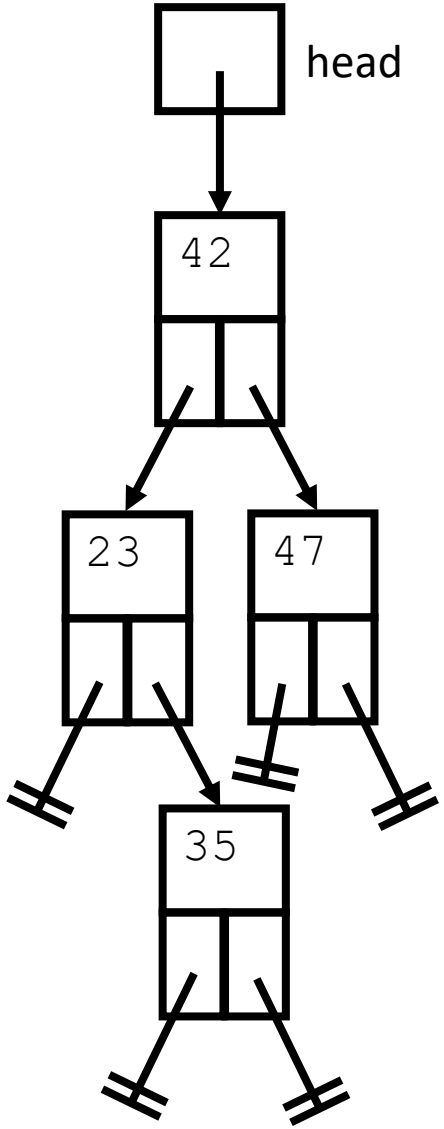
```
Procedure Search(
 cur iot in Ptr toa Node,
 target iot in num)
 if(cur = NIL) then
  print("Not Found")
 elseif(cur^.data = target)
  print("Target Found")
 elseif(cur^.data > target)
  Search(cur^.left, target)
 else
  Search(cur^.right, target)
 endif
endprocedure // Search
```



```
Procedure Search(
 cur iot in Ptr toa Node,
 target iot in num)
 if(cur = NIL) then
  print("Not Found")
 elseif(cur^.data = target)
  print("Target Found")
 elseif(cur^.data > target)
  Search(cur^.left, target)
 else
  Search(cur^.right, target)
 endif
endprocedure // Search
```

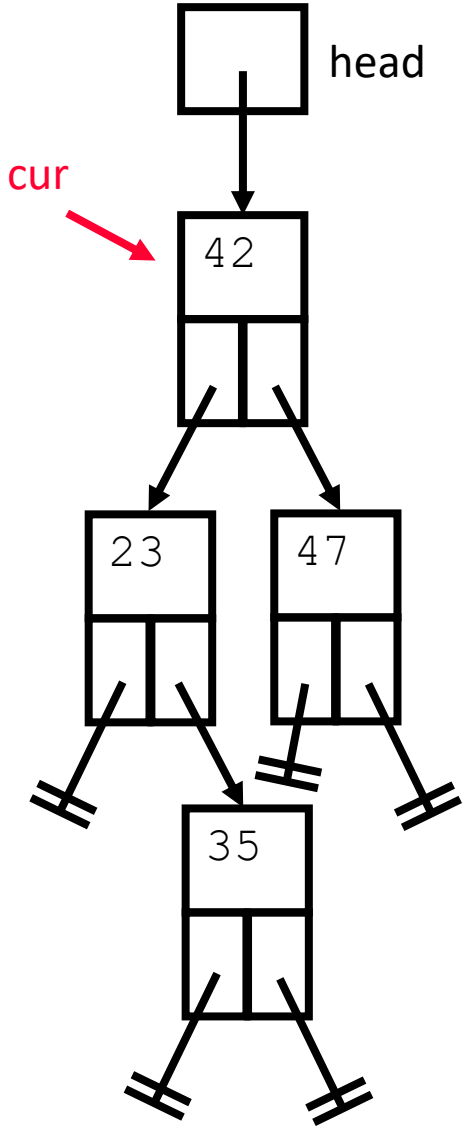


```
.
.
Search(head, 35)
Search(head, 87)
.
.
```

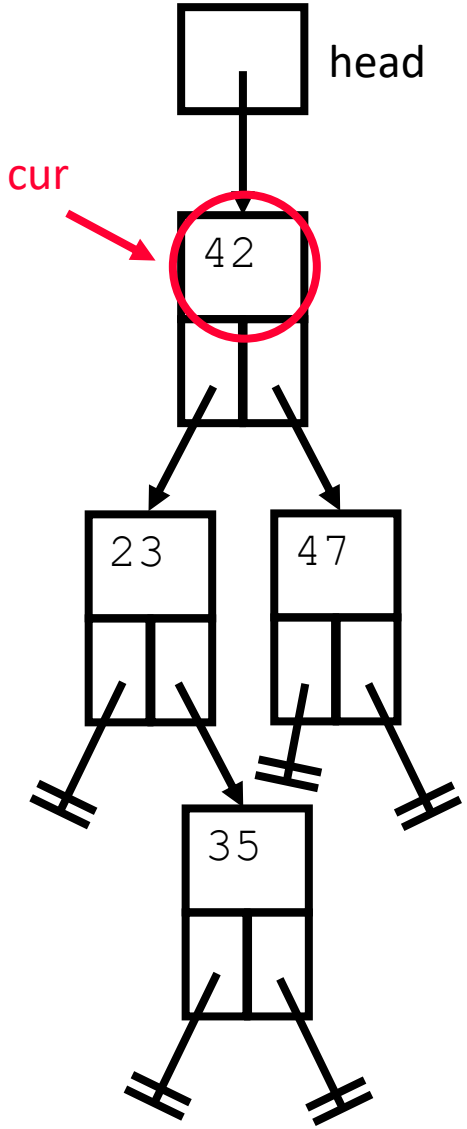


```
Procedure Search(
 cur iot in Ptr toa Node,
 target iot in num)
 if(cur = NIL) then
  print("Not Found")
 elseif(cur^.data = target)
  print("Target Found")
 elseif(cur^.data > target)
  Search(cur^.left, target)
 else
  Search(cur^.right, target)
 endif
endprocedure // Search
```

```
target = 87
```

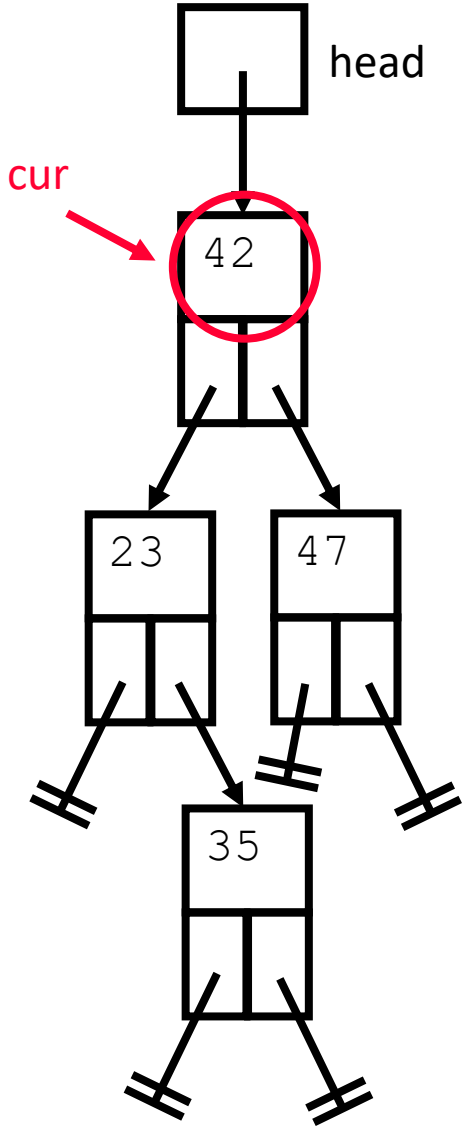


```
Procedure Search(
 cur iot in Ptr toa Node,
 target iot in num)
 if(cur = NIL) then
  print("Not Found")
 elseif(cur^.data = target)
  print("Target Found")
 elseif(cur^.data > target)
  Search(cur^.left, target)
 else
  Search(cur^.right, target)
 endif
endprocedure // Search
```



target = 87

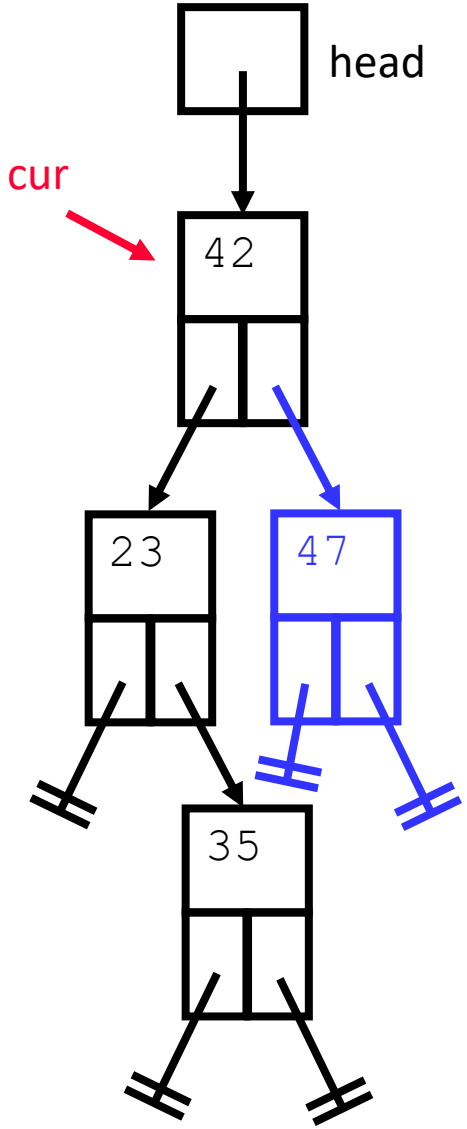
```
Procedure Search(
 cur iot in Ptr toa Node,
 target iot in num)
 if(cur = NIL) then
  print("Not Found")
 elseif(cur^.data = target)
  print("Target Found")
 elseif(cur^.data > target)
  Search(cur^.left, target)
 else
  Search(cur^.right, target)
 endif
endprocedure // Search
```



target = 87

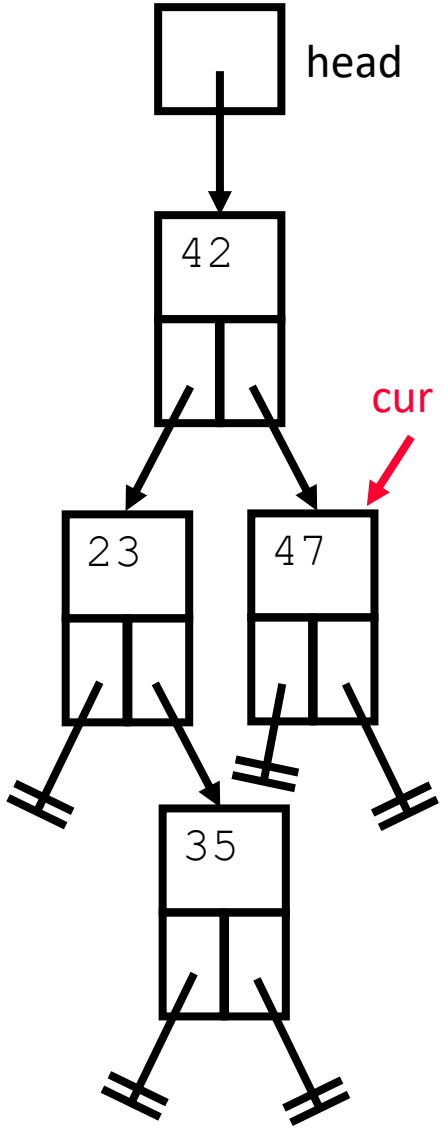
```
Procedure Search(
 cur iot in Ptr toa Node,
 target iot in num)
 if(cur = NIL) then
  print("Not Found")
 elseif(cur^.data = target)
  print("Target Found")
 elseif(cur^.data > target)
  Search(cur^.left, target)
 else
  Search(cur^.right, target)
 endif
endprocedure // Search
```

```
target = 87
```



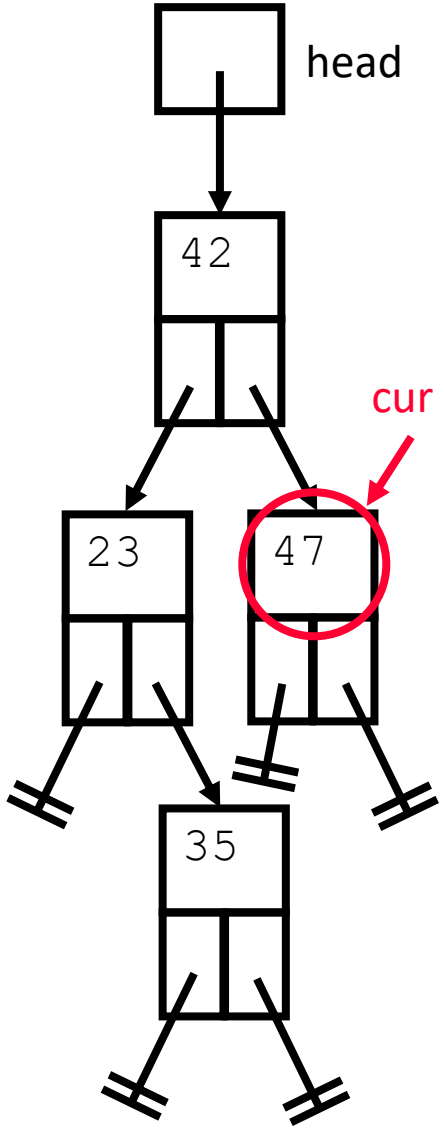
```
Procedure Search(
 cur iot in Ptr toa Node,
 target iot in num)
 if(cur = NIL) then
  print("Not Found")
 elseif(cur^.data = target)
  print("Target Found")
 elseif(cur^.data > target)
  Search(cur^.left, target)
 else
  Search(cur^.right, target)
 endif
endprocedure // Search
```

```
target = 87
```



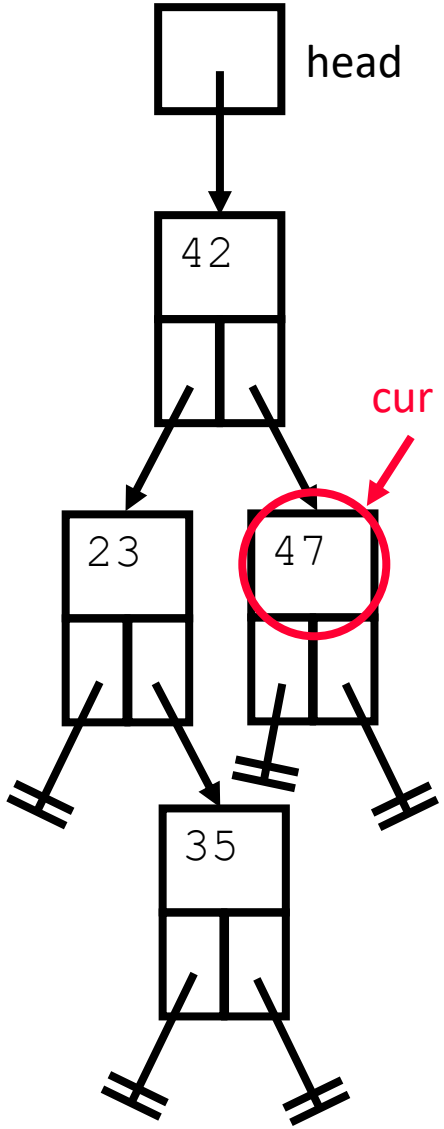
```
Procedure Search(
 cur iot in Ptr toa Node,
 target iot in num)
 if(cur = NIL) then
  print("Not Found")
 elseif(cur^.data = target)
  print("Target Found")
 elseif(cur^.data > target)
  Search(cur^.left, target)
 else
  Search(cur^.right, target)
 endif
endprocedure // Search
```

```
target = 87
```



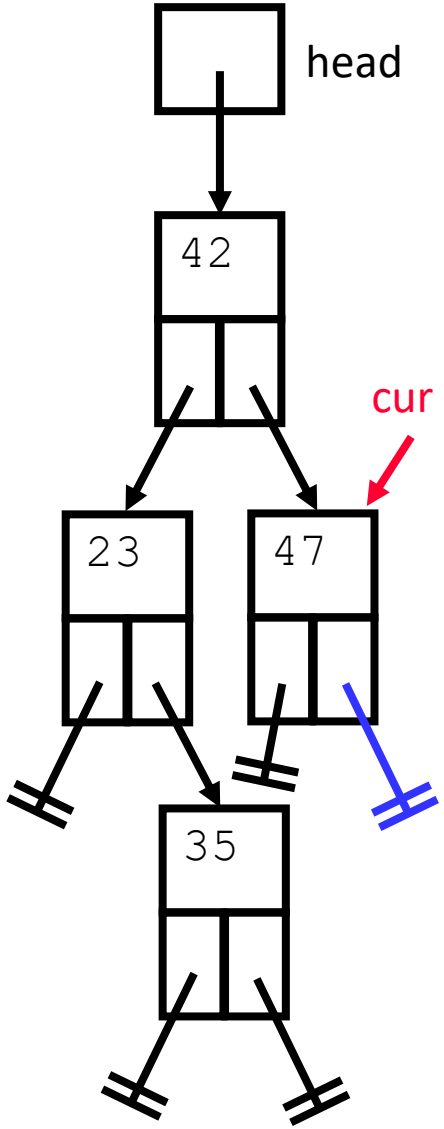
```
Procedure Search(
 cur iot in Ptr toa Node,
 target iot in num)
 if(cur = NIL) then
  print("Not Found")
 elseif(cur^.data = target)
  print("Target Found")
 elseif(cur^.data > target)
  Search(cur^.left, target)
 else
  Search(cur^.right, target)
 endif
endprocedure // Search
```

```
target = 87
```



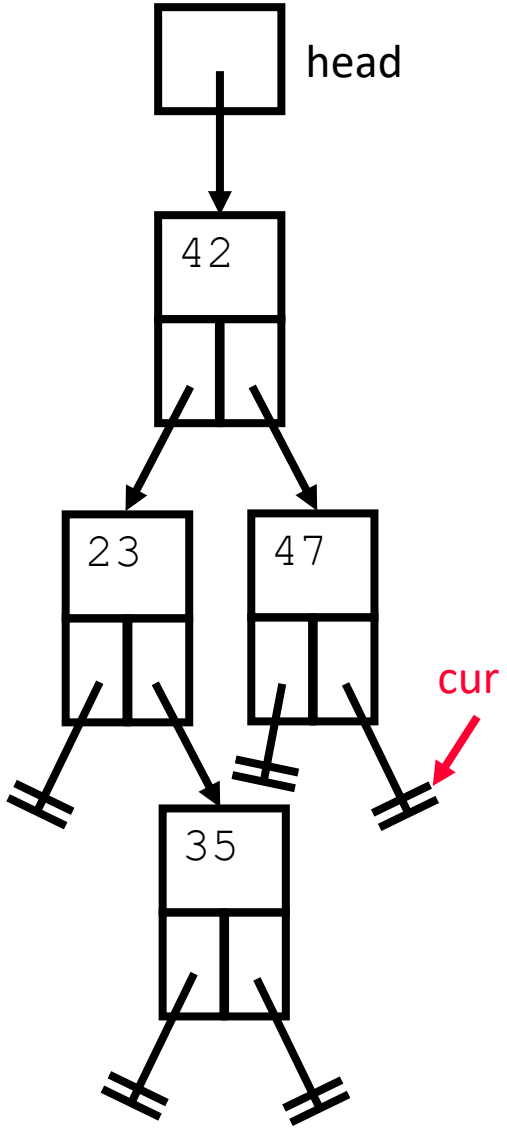
```
Procedure Search(
 cur iot in Ptr toa Node,
 target iot in num)
 if(cur = NIL) then
  print("Not Found")
 elseif(cur^.data = target)
  print("Target Found")
 elseif(cur^.data > target)
  Search(cur^.left, target)
 else
  Search(cur^.right, target)
 endif
endprocedure // Search
```

```
target = 87
```



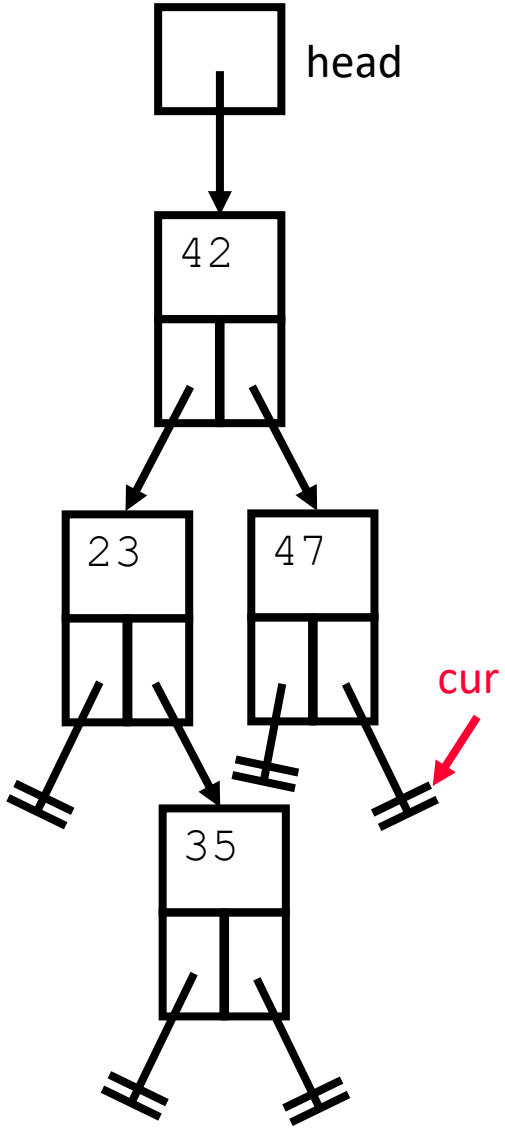
```
Procedure Search(
 cur iot in Ptr toa Node,
 target iot in num)
 if(cur = NIL) then
  print("Not Found")
 elseif(cur^.data = target)
  print("Target Found")
 elseif(cur^.data > target)
  Search(cur^.left, target)
 else
  Search(cur^.right, target)
 endif
endprocedure // Search
```

```
target = 87
```

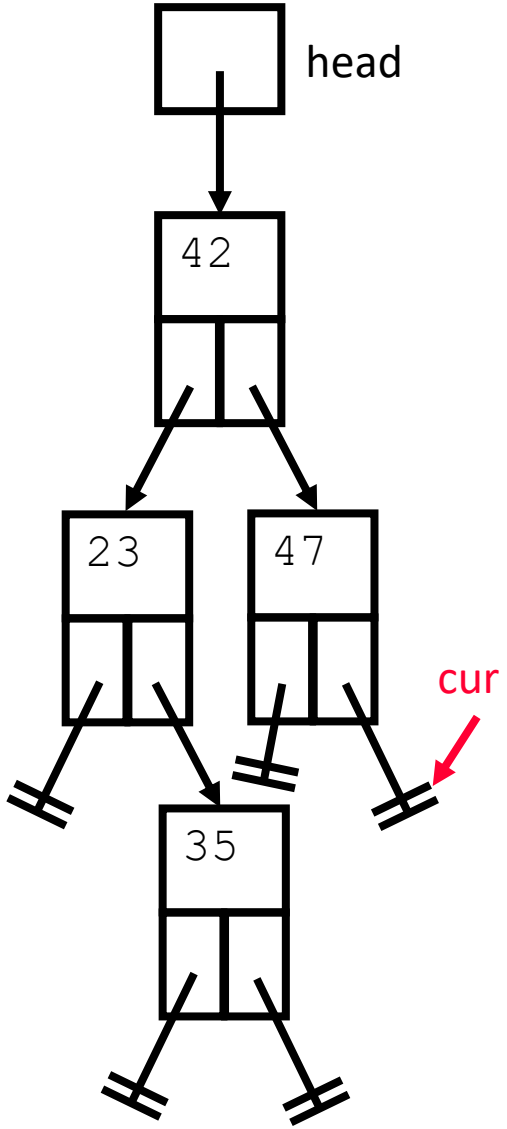


```
Procedure Search(
 cur iot in Ptr toa Node,
 target iot in num)
 if(cur = NIL) then
  print("Not Found")
 elseif(cur^.data = target)
  print("Target Found")
 elseif(cur^.data > target)
  Search(cur^.left, target)
 else
  Search(cur^.right, target)
 endif
endprocedure // Search
```

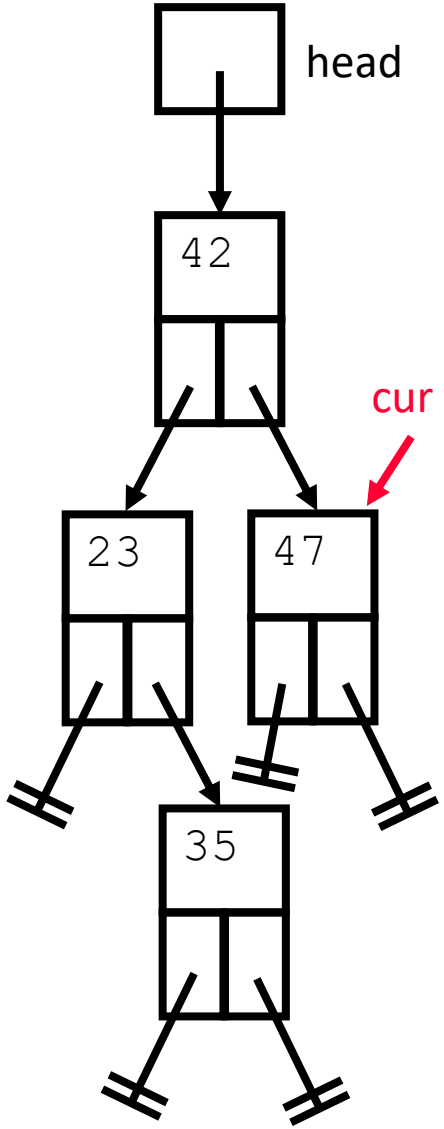
Not Found



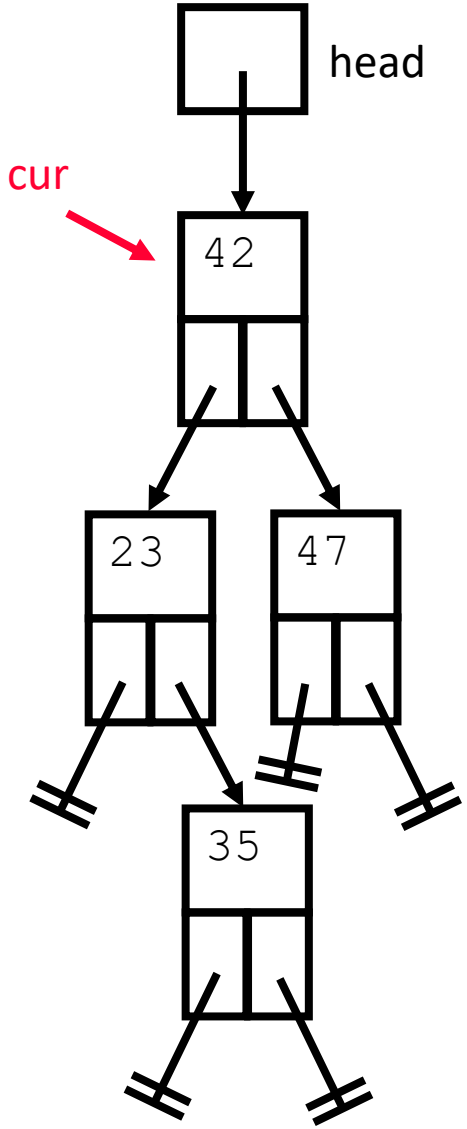
```
Procedure Search(
 cur iot in Ptr toa Node,
 target iot in num)
 if(cur = NIL) then
  print("Not Found")
 elseif(cur^.data = target)
  print("Target Found")
 elseif(cur^.data > target)
  Search(cur^.left, target)
 else
  Search(cur^.right, target)
 endif
endprocedure // Search
```



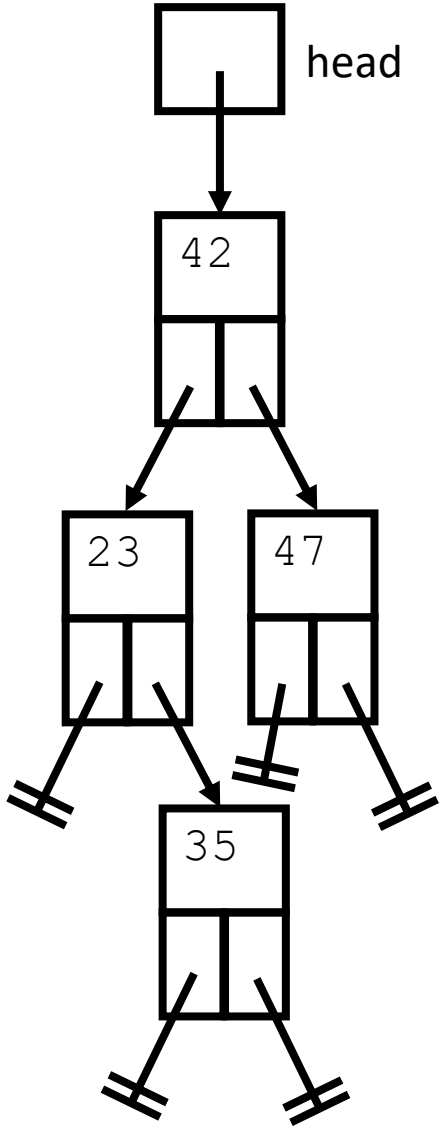
```
Procedure Search(
 cur iot in Ptr toa Node,
 target iot in num)
 if(cur = NIL) then
  print("Not Found")
 elseif(cur^.data = target)
  print("Target Found")
 elseif(cur^.data > target)
  Search(cur^.left, target)
 else
  Search(cur^.right, target)
 endif
endprocedure // Search
```



```
Procedure Search(
 cur iot in Ptr toa Node,
 target iot in num)
 if(cur = NIL) then
  print("Not Found")
 elseif(cur^.data = target)
  print("Target Found")
 elseif(cur^.data > target)
  Search(cur^.left, target)
 else
  Search(cur^.right, target)
 endif
endprocedure // Search
```



```
.
.
Search(head, 35)
Search(head, 87)
.
.
```



Summary

- **We can cut the work in half after each comparison.**
- **Recurse in one direction – left or right.**
- **When we reach NIL, then we no the value wasn't in the binary search tree.**

ADDING A NODE

Adding the Node

- **Current is an in/out pointer**
  - **We need information IN to evaluate current**
  - **We need to send information OUT because we're changing the tree (adding a node)**

- **Once we've found the correct location:**
  - **Create a new node**
  - **Fill in the data field (with the new value to add)**
  - **Make the left and right pointers point to nil (to cleanly terminate the tree)**

Adding the Node

```
current <- new(Node)
current^.data <- value_to_add
current^.left <- nil
current^.right <- nil
```

The Entire Module

```
procedure Insert(cur iot in/out Ptr toa Node,
                 data_in iot in num)
  if(cur = NIL) then
    cur <- new(Node)
    cur^.data <- data_in
    cur^.left <- NIL
    cur^.right <- NIL
  elseif(cur^.data > data_in)
    Insert(cur^.left, data_in)
  else
    Insert(cur^.right, data_in)
  endif
endprocedure // Insert
```

Tracing Example

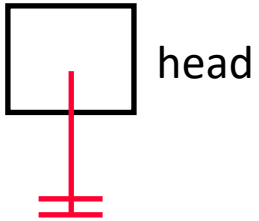
The following example shows a trace of the BST insert.

- **Begin with an empty BST (a pointer)**
- **Add elements 42, 23, 35, 47 in the correct positions.**

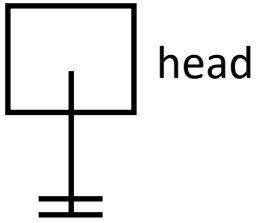
```
Head iot Ptr toa Node
head <- NIL
Insert(head, 42)
```

```
Head iot Ptr toa Node
head <- NIL
Insert(head, 42)
```

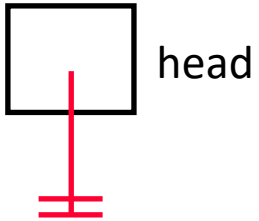
```
Head iot Ptr toa Node
head <- NIL
Insert(head, 42)
```



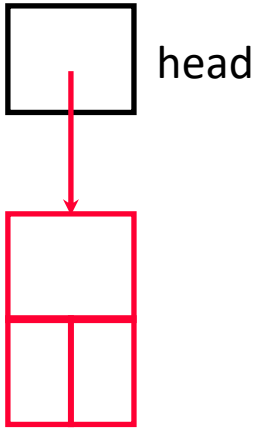
```
Head iot Ptr toa Node
head <- NIL
Insert(head, 42)
```



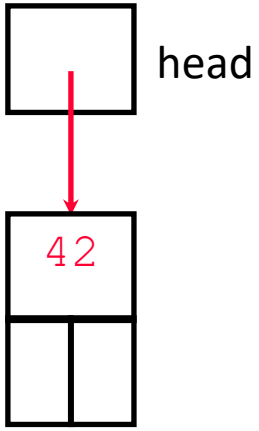
```
procedure Insert(
 cur iot in/out Ptr toa Node,
 data_in iot in num)
 if(cur = NIL) then
  cur <- new(Node)
  cur^.data <- data_in
  cur^.left <- NIL
  cur^.right <- NIL
 elseif(cur^.data > data_in)
  Insert(cur^.left, data_in)
 else
  Insert(cur^.right, data_in)
 endif
endprocedure // Insert
```



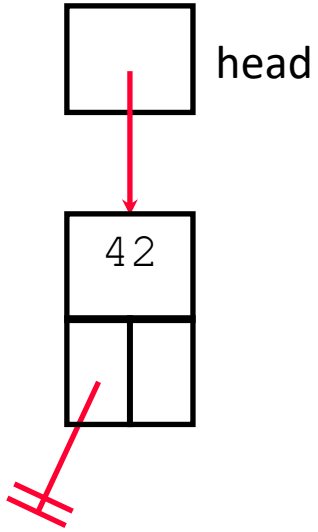
```
procedure Insert(
 cur iot in/out Ptr toa Node,
 data_in iot in num)
 if(cur = NIL) then
  cur <- new(Node)
  cur^.data <- data_in
  cur^.left <- NIL
  cur^.right <- NIL
 elseif(cur^.data > data_in)
  Insert(cur^.left, data_in)
 else
  Insert(cur^.right, data_in)
 endif
endprocedure // Insert
```



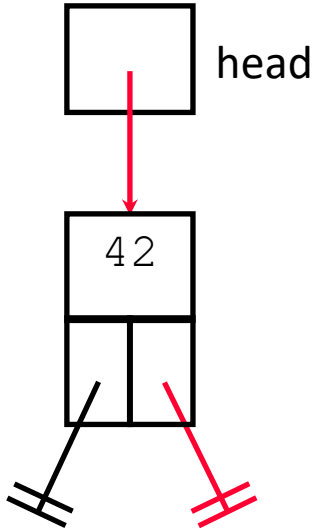
```
procedure Insert(
 cur iot in/out Ptr toa Node,
 data_in iot in num)
 if(cur = NIL) then
  cur <- new(Node)
  cur^.data <- data_in
  cur^.left <- NIL
  cur^.right <- NIL
 elseif(cur^.data > data_in)
  Insert(cur^.left, data_in)
 else
  Insert(cur^.right, data_in)
 endif
endprocedure // Insert
```



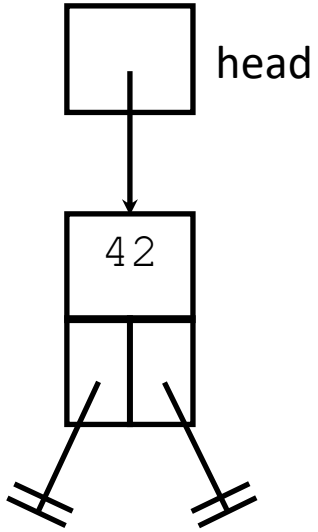
```
procedure Insert(
 cur iot in/out Ptr toa Node,
 data_in iot in num)
 if(cur = NIL) then
  cur <- new(Node)
  cur^.data <- data_in
  cur^.left <- NIL
  cur^.right <- NIL
 elseif(cur^.data > data_in)
  Insert(cur^.left, data_in)
 else
  Insert(cur^.right, data_in)
 endif
endprocedure // Insert
```



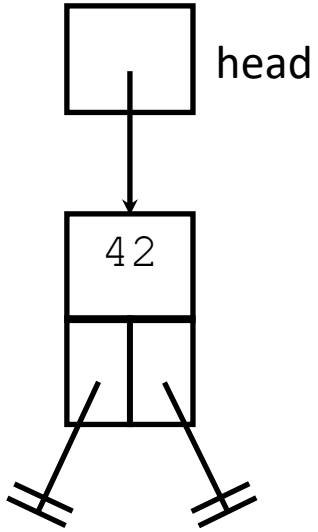
```
procedure Insert(
 cur iot in/out Ptr toa Node,
 data_in iot in num)
 if(cur = NIL) then
  cur <- new(Node)
  cur^.data <- data_in
  cur^.left <- NIL
  cur^.right <- NIL
 elseif(cur^.data > data_in)
  Insert(cur^.left, data_in)
 else
  Insert(cur^.right, data_in)
 endif
endprocedure // Insert
```



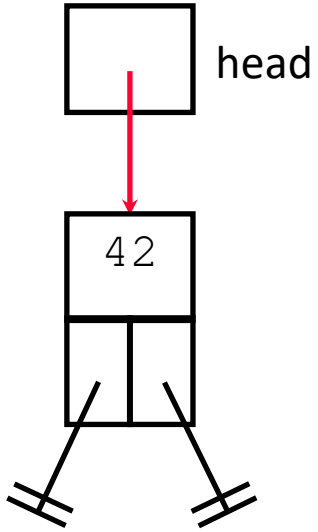
```
procedure Insert(
 cur iot in/out Ptr toa Node,
 data_in iot in num)
 if(cur = NIL) then
  cur <- new(Node)
  cur^.data <- data_in
  cur^.left <- NIL
  cur^.right <- NIL
 elseif(cur^.data > data_in)
  Insert(cur^.left, data_in)
 else
  Insert(cur^.right, data_in)
 endif
endprocedure // Insert
```



```
.
.
Insert(head, 23)
Insert(head, 35)
Insert(head, 47)
.
.
```

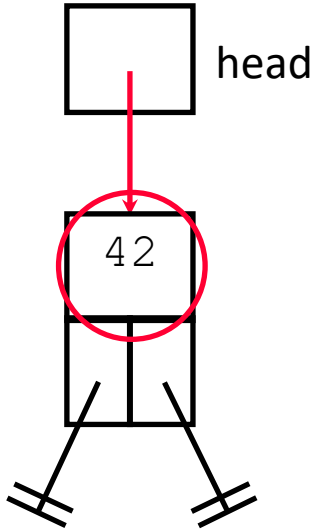


```
procedure Insert(
 cur iot in/out Ptr toa Node,
 data_in iot in num)
 if(cur = NIL) then
  cur <- new(Node)
  cur^.data <- data_in
  cur^.left <- NIL
  cur^.right <- NIL
 elseif(cur^.data > data_in)
  Insert(cur^.left, data_in)
 else
  Insert(cur^.right, data_in)
 endif
endprocedure // Insert
```



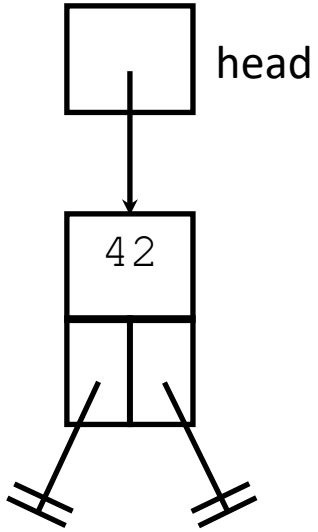
data_in = 23

```
procedure Insert(
 cur iot in/out Ptr toa Node,
 data_in iot in num)
 if(cur = NIL) then
  cur <- new(Node)
  cur^.data <- data_in
  cur^.left <- NIL
  cur^.right <- NIL
 elseif(cur^.data > data_in)
  Insert(cur^.left, data_in)
 else
  Insert(cur^.right, data_in)
 endif
endprocedure // Insert
```




```
data_in = 23
```

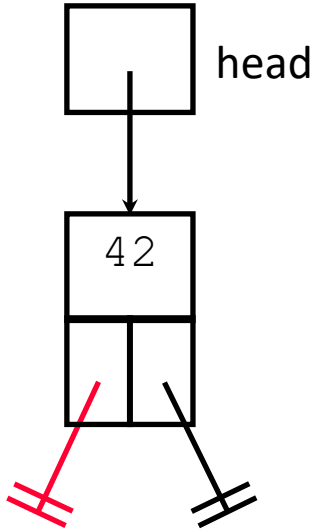
```
procedure Insert(
 cur iot in/out Ptr toa Node,
 data_in iot in num)
 if(cur = NIL) then
  cur <- new(Node)
  cur^.data <- data_in
  cur^.left <- NIL
  cur^.right <- NIL
 elseif(cur^.data > data_in)
  Insert(cur^.left, data_in)
 else
  Insert(cur^.right, data_in)
 endif
endprocedure // Insert
```




```
data_in = 23
```

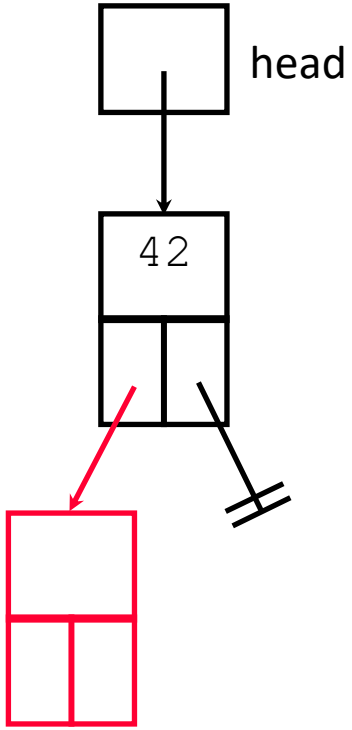
```
procedure Insert(
 cur iot in/out Ptr toa Node,
 data_in iot in num)
 if(cur = NIL) then
  cur <- new(Node)
  cur^.data <- data_in
  cur^.left <- NIL
  cur^.right <- NIL
 elseif(cur^.data > data_in)
  Insert(cur^.left, data_in)
 else
  Insert(cur^.right, data_in)
 endif
endprocedure // Insert
```

data_in = 23



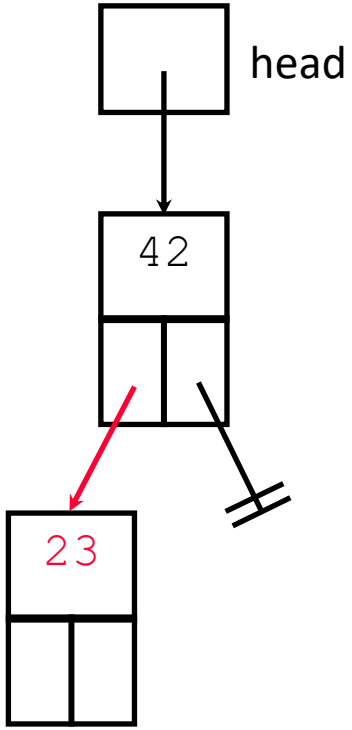
```
procedure Insert(
 cur iot in/out Ptr toa Node,
 data_in iot in num)
 if(cur = NIL) then
  cur <- new(Node)
  cur^.data <- data_in
  cur^.left <- NIL
  cur^.right <- NIL
 elseif(cur^.data > data_in)
  Insert(cur^.left, data_in)
 else
  Insert(cur^.right, data_in)
 endif
endprocedure // Insert
```

data_in = 23



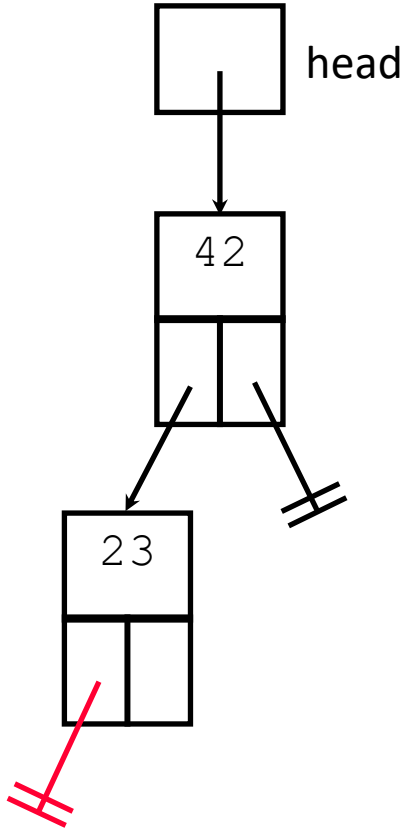
```
procedure Insert(
 cur iot in/out Ptr toa Node,
 data_in iot in num)
 if(cur = NIL) then
  cur <- new(Node)
  cur^.data <- data_in
  cur^.left <- NIL
  cur^.right <- NIL
 elseif(cur^.data > data_in)
  Insert(cur^.left, data_in)
 else
  Insert(cur^.right, data_in)
 endif
endprocedure // Insert
```

data_in = 23



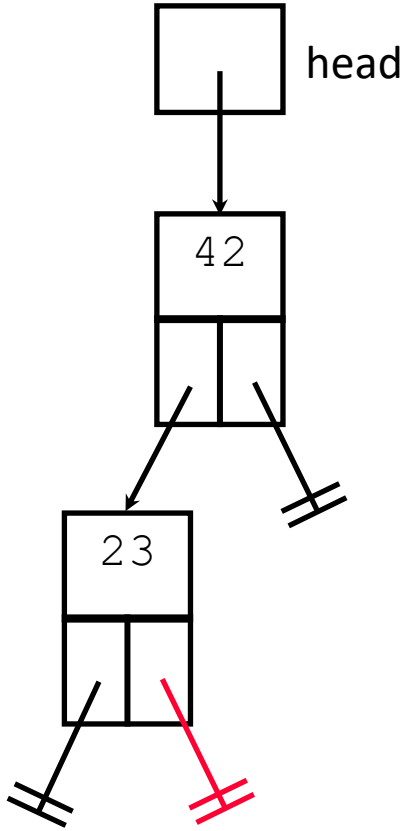
```
procedure Insert(
 cur iot in/out Ptr toa Node,
 data_in iot in num)
 if(cur = NIL) then
  cur <- new(Node)
  cur^.data <- data_in
  cur^.left <- NIL
  cur^.right <- NIL
 elseif(cur^.data > data_in)
  Insert(cur^.left, data_in)
 else
  Insert(cur^.right, data_in)
 endif
endprocedure // Insert
```

data_in = 23



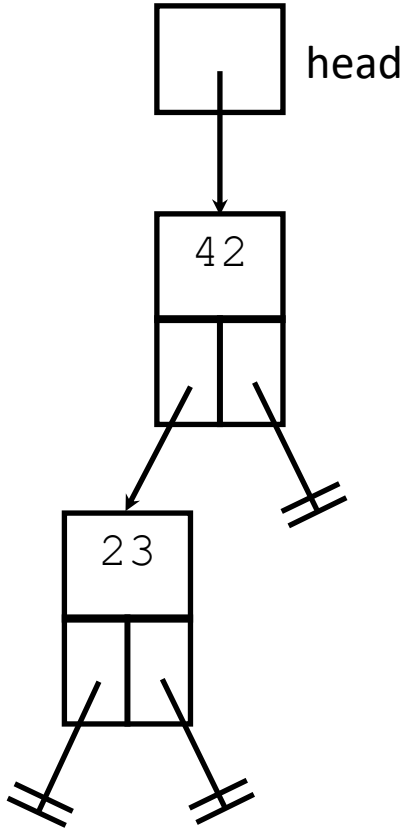
```
procedure Insert(
 cur iot in/out Ptr toa Node,
 data_in iot in num)
 if(cur = NIL) then
  cur <- new(Node)
  cur^.data <- data_in
  cur^.left <- NIL
  cur^.right <- NIL
 elseif(cur^.data > data_in)
  Insert(cur^.left, data_in)
 else
  Insert(cur^.right, data_in)
 endif
endprocedure // Insert
```

data_in = 23

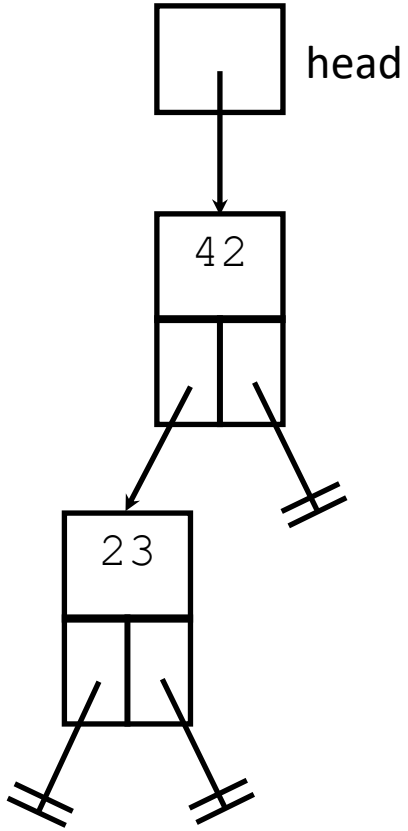


```
procedure Insert(
 cur iot in/out Ptr toa Node,
 data_in iot in num)
 if(cur = NIL) then
  cur <- new(Node)
  cur^.data <- data_in
  cur^.left <- NIL
  cur^.right <- NIL
 elseif(cur^.data > data_in)
  Insert(cur^.left, data_in)
 else
  Insert(cur^.right, data_in)
 endif
endprocedure // Insert
```

data_in = 23

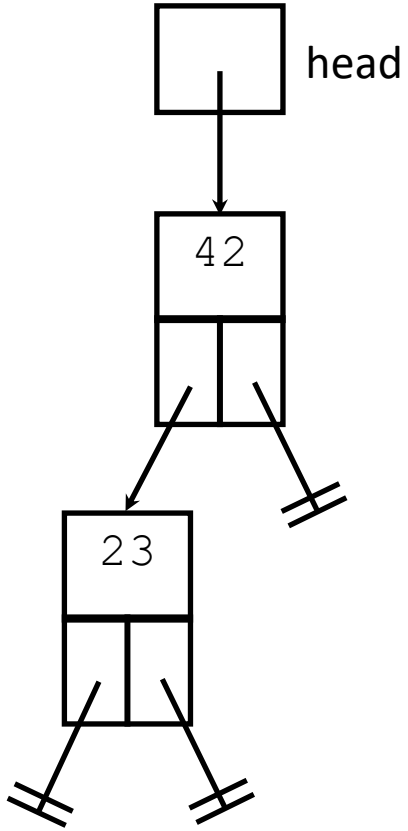


```
procedure Insert(
 cur iot in/out Ptr toa Node,
 data_in iot in num)
 if(cur = NIL) then
  cur <- new(Node)
  cur^.data <- data_in
  cur^.left <- NIL
  cur^.right <- NIL
 elseif(cur^.data > data_in)
  Insert(cur^.left, data_in)
 else
  Insert(cur^.right, data_in)
 endif
endprocedure // Insert
```

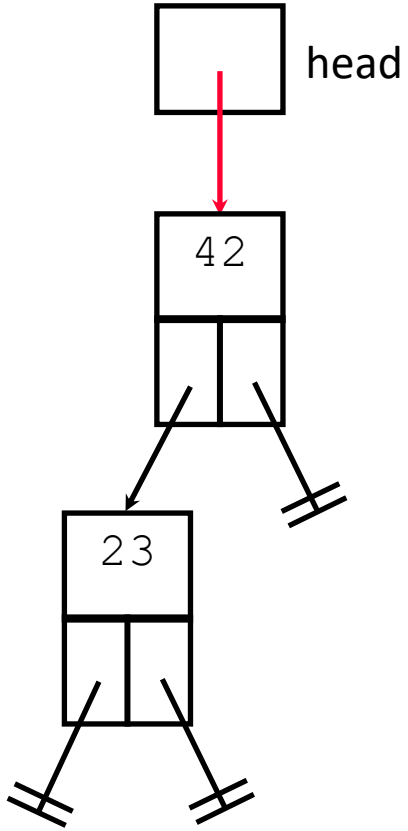


data_in = 23

```
.
.
Insert(head, 23)
Insert(head, 35)
Insert(head, 47)
.
.
```

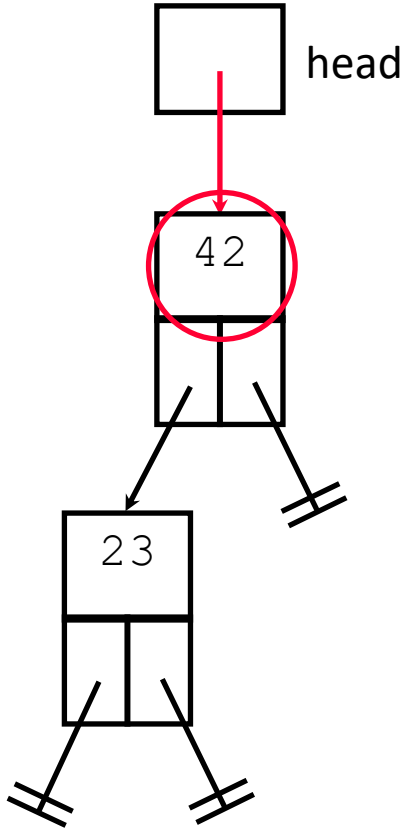


```
procedure Insert(
 cur iot in/out Ptr toa Node,
 data_in iot in num)
 if(cur = NIL) then
  cur <- new(Node)
  cur^.data <- data_in
  cur^.left <- NIL
  cur^.right <- NIL
 elseif(cur^.data > data_in)
  Insert(cur^.left, data_in)
 else
  Insert(cur^.right, data_in)
 endif
endprocedure // Insert
```



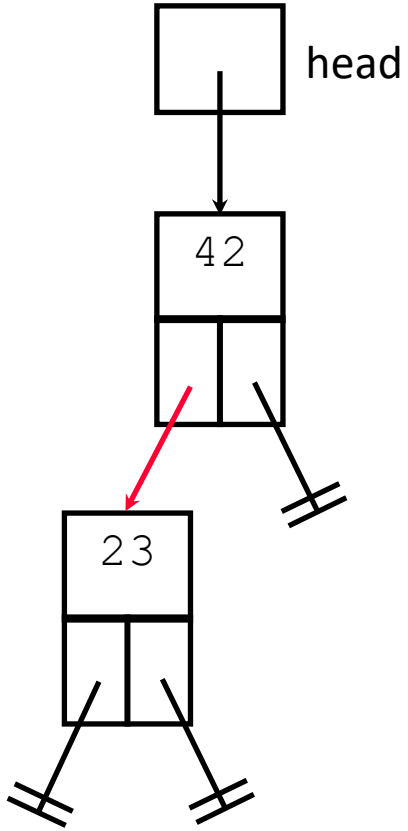

```
data_in = 35
```

```
procedure Insert(
 cur iot in/out Ptr toa Node,
 data_in iot in num)
 if(cur = NIL) then
  cur <- new(Node)
  cur^.data <- data_in
  cur^.left <- NIL
  cur^.right <- NIL
 elseif(cur^.data > data_in)
  Insert(cur^.left, data_in)
 else
  Insert(cur^.right, data_in)
 endif
endprocedure // Insert
```



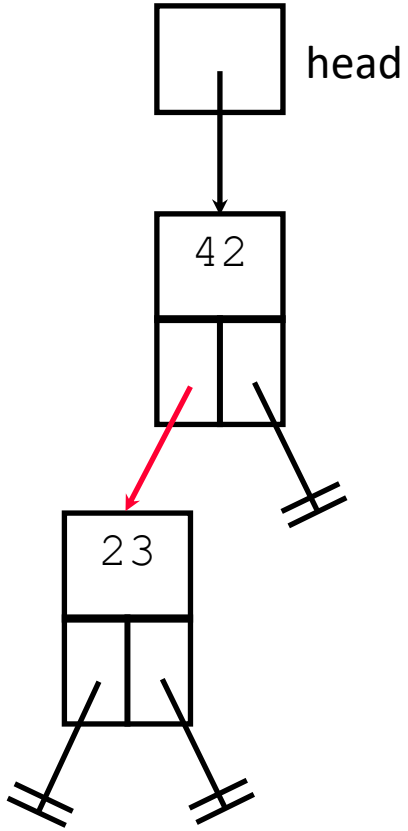

```
data_in = 35
```

```
procedure Insert(
 cur iot in/out Ptr toa Node,
 data_in iot in num)
 if(cur = NIL) then
  cur <- new(Node)
  cur^.data <- data_in
  cur^.left <- NIL
  cur^.right <- NIL
 elseif(cur^.data > data_in)
  Insert(cur^.left, data_in)
 else
  Insert(cur^.right, data_in)
 endif
endprocedure // Insert
```



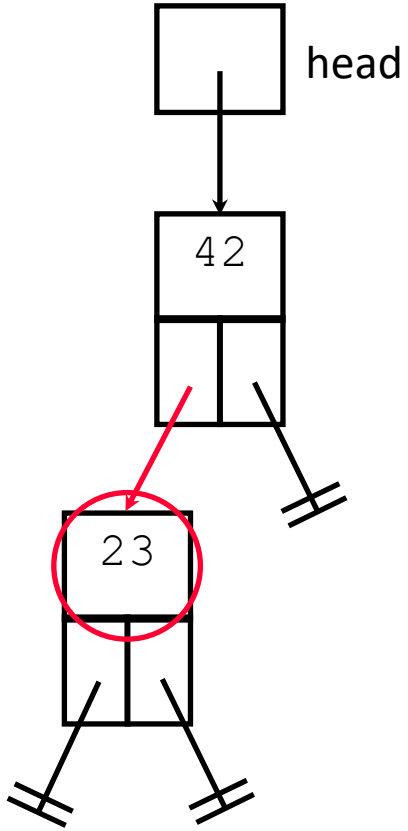

```
data_in = 35
```

```
procedure Insert(
 cur iot in/out Ptr toa Node,
 data_in iot in num)
 if(cur = NIL) then
  cur <- new(Node)
  cur^.data <- data_in
  cur^.left <- NIL
  cur^.right <- NIL
 elseif(cur^.data > data_in)
  Insert(cur^.left, data_in)
 else
  Insert(cur^.right, data_in)
 endif
endprocedure // Insert
```



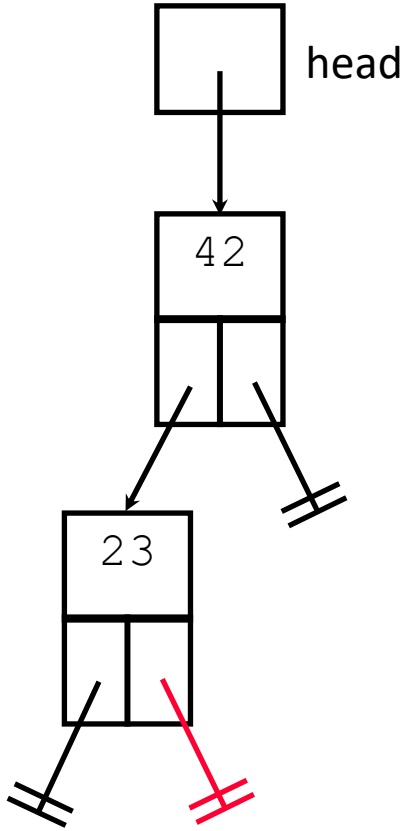
data_in = 35

```
procedure Insert(
 cur iot in/out Ptr toa Node,
 data_in iot in num)
 if(cur = NIL) then
  cur <- new(Node)
  cur^.data <- data_in
  cur^.left <- NIL
  cur^.right <- NIL
 elseif(cur^.data > data_in)
  Insert(cur^.left, data_in)
 else
  Insert(cur^.right, data_in)
 endif
endprocedure // Insert
```



data_in = 35

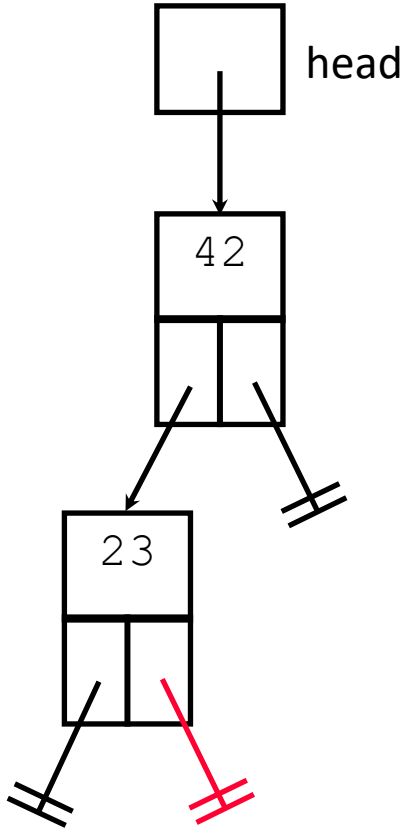
```
procedure Insert(
 cur iot in/out Ptr toa Node,
 data_in iot in num)
 if(cur = NIL) then
  cur <- new(Node)
  cur^.data <- data_in
  cur^.left <- NIL
  cur^.right <- NIL
 elseif(cur^.data > data_in)
  Insert(cur^.left, data_in)
 else
  Insert(cur^.right, data_in)
 endif
endprocedure // Insert
```



data_in = 35

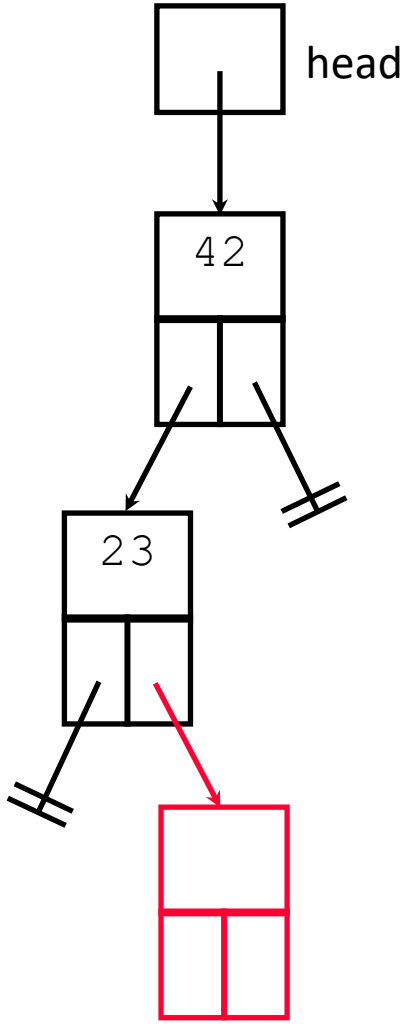
```
procedure Insert(
 cur iot in/out Ptr toa Node,
 data_in iot in num)
 if(cur = NIL) then
  cur <- new(Node)
  cur^.data <- data_in
  cur^.left <- NIL
  cur^.right <- NIL
 elseif(cur^.data > data_in)
  Insert(cur^.left, data_in)
 else
  Insert(cur^.right, data_in)
 endif
endprocedure // Insert
```

data_in = 35

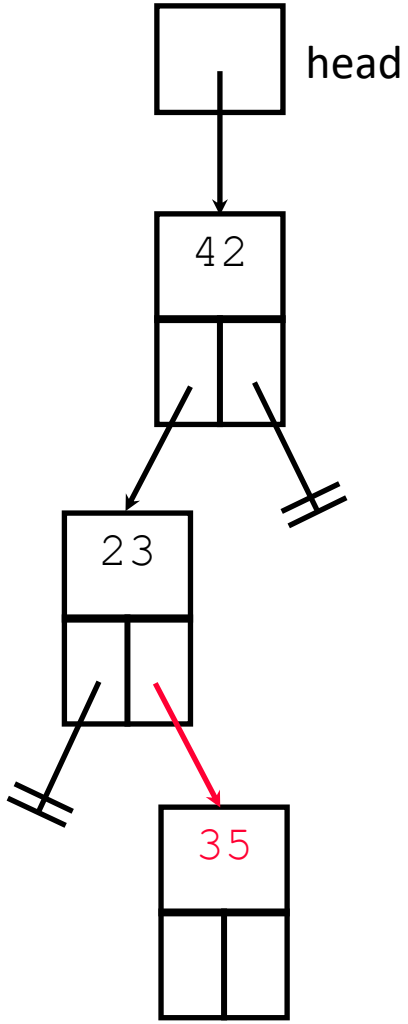


```
procedure Insert(
 cur iot in/out Ptr toa Node,
 data_in iot in num)
 if(cur = NIL) then
  cur <- new(Node)
  cur^.data <- data_in
  cur^.left <- NIL
  cur^.right <- NIL
 elseif(cur^.data > data_in)
  Insert(cur^.left, data_in)
 else
  Insert(cur^.right, data_in)
 endif
endprocedure // Insert
```

data_in = 35



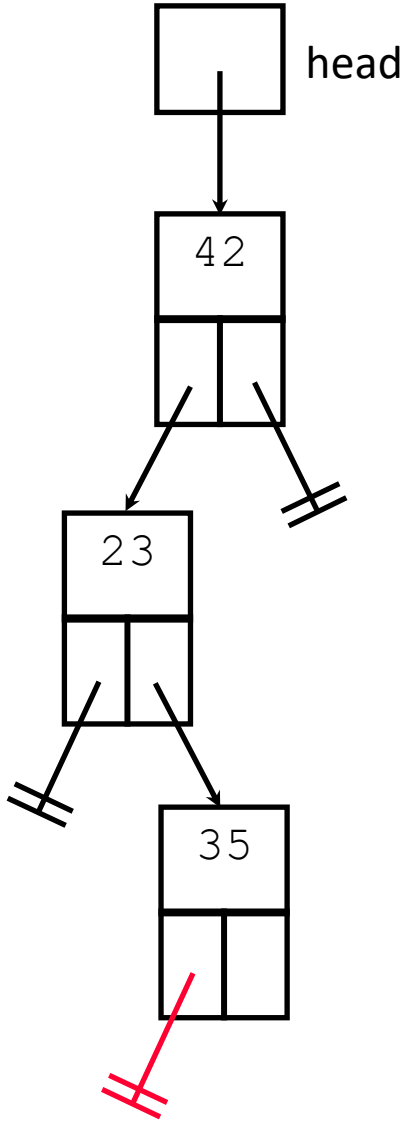
```
procedure Insert(
 cur iot in/out Ptr toa Node,
 data_in iot in num)
 if(cur = NIL) then
  cur <- new(Node)
  cur^.data <- data_in
  cur^.left <- NIL
  cur^.right <- NIL
 elseif(cur^.data > data_in)
  Insert(cur^.left, data_in)
 else
  Insert(cur^.right, data_in)
 endif
endprocedure // Insert
```



data_in = 35

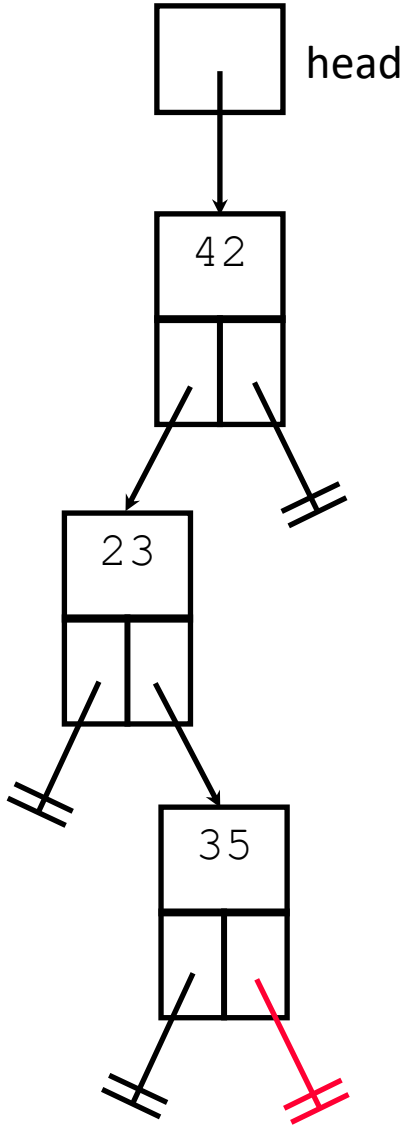
```
procedure Insert(
 cur iot in/out Ptr toa Node,
 data_in iot in num)
 if(cur = NIL) then
  cur <- new(Node)
  cur^.data <- data_in
  cur^.left <- NIL
  cur^.right <- NIL
 elseif(cur^.data > data_in)
  Insert(cur^.left, data_in)
 else
  Insert(cur^.right, data_in)
 endif
endprocedure // Insert
```

data_in = 35



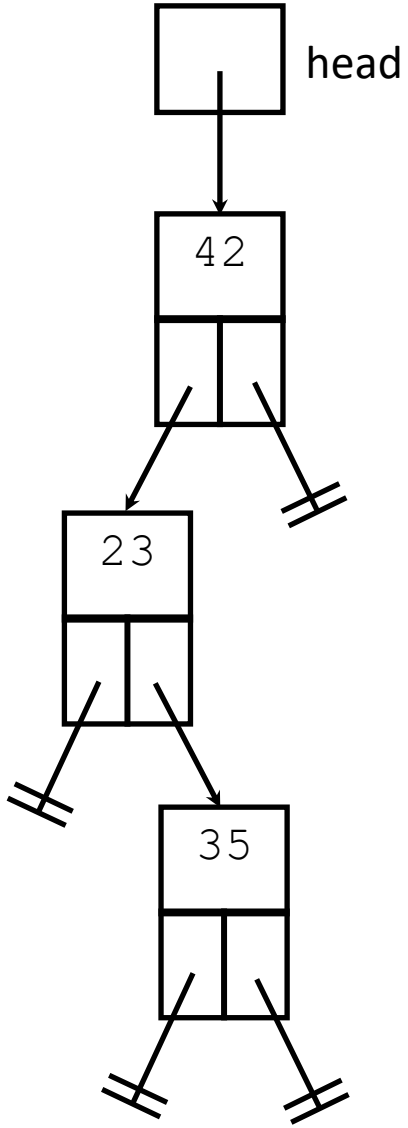
```
procedure Insert(
 cur iot in/out Ptr toa Node,
 data_in iot in num)
 if(cur = NIL) then
  cur <- new(Node)
  cur^.data <- data_in
  cur^.left <- NIL
  cur^.right <- NIL
 elseif(cur^.data > data_in)
  Insert(cur^.left, data_in)
 else
  Insert(cur^.right, data_in)
 endif
endprocedure // Insert
```

data_in = 35



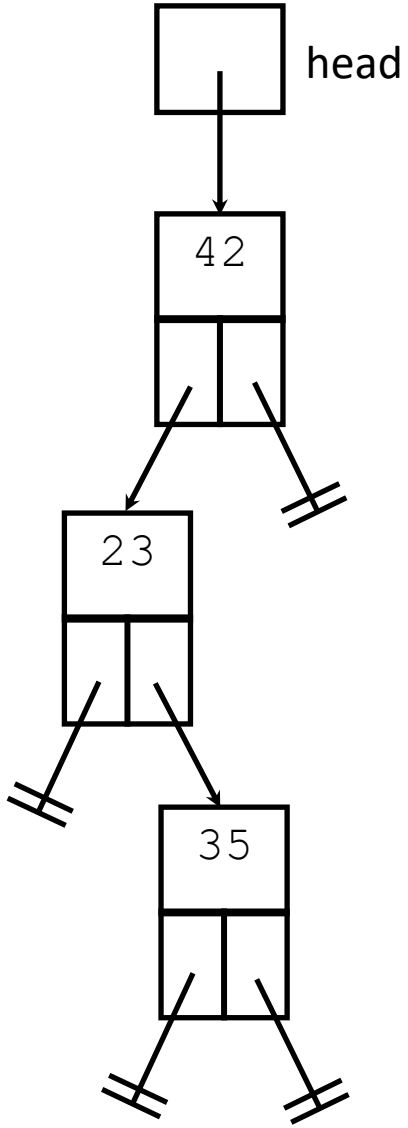
```
procedure Insert(
 cur iot in/out Ptr toa Node,
 data_in iot in num)
 if(cur = NIL) then
  cur <- new(Node)
  cur^.data <- data_in
  cur^.left <- NIL
  cur^.right <- NIL
 elseif(cur^.data > data_in)
  Insert(cur^.left, data_in)
 else
  Insert(cur^.right, data_in)
 endif
endprocedure // Insert
```

data_in = 35

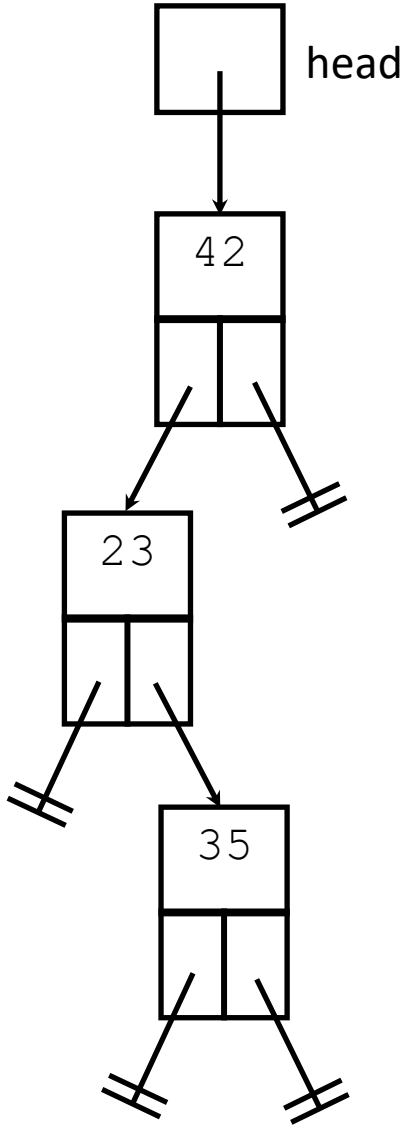


```
procedure Insert(
 cur iot in/out Ptr toa Node,
 data_in iot in num)
 if(cur = NIL) then
  cur <- new(Node)
  cur^.data <- data_in
  cur^.left <- NIL
  cur^.right <- NIL
 elseif(cur^.data > data_in)
  Insert(cur^.left, data_in)
 else
  Insert(cur^.right, data_in)
 endif
endprocedure // Insert
```

data_in = 35

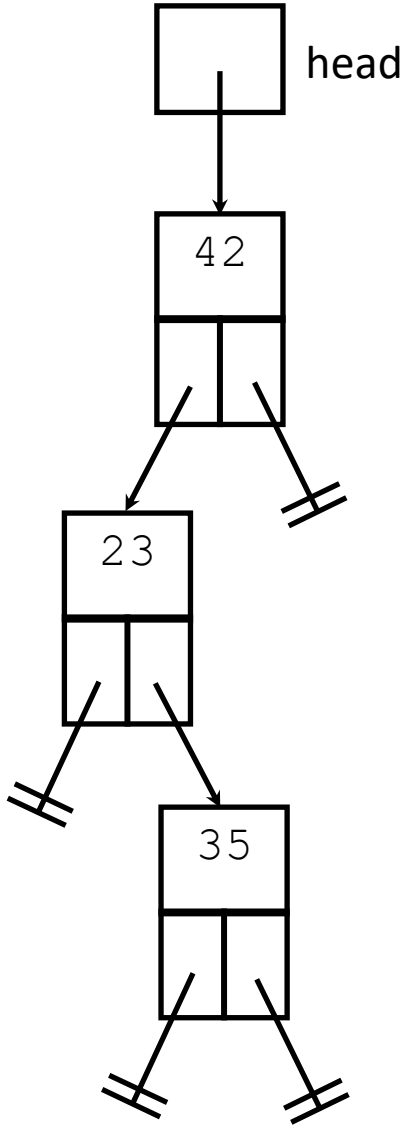


```
procedure Insert(
 cur iot in/out Ptr toa Node,
 data_in iot in num)
 if(cur = NIL) then
  cur <- new(Node)
  cur^.data <- data_in
  cur^.left <- NIL
  cur^.right <- NIL
 elseif(cur^.data > data_in)
  Insert(cur^.left, data_in)
 else
  Insert(cur^.right, data_in)
 endif
endprocedure // Insert
```

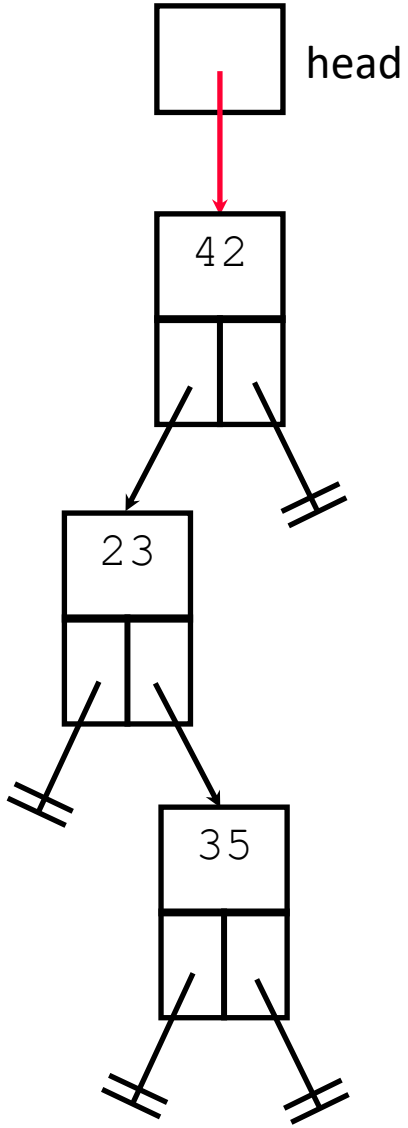


data_in = 35

```
.
.
Insert(head, 23)
Insert(head, 35)
Insert(head, 47)
.
.
```



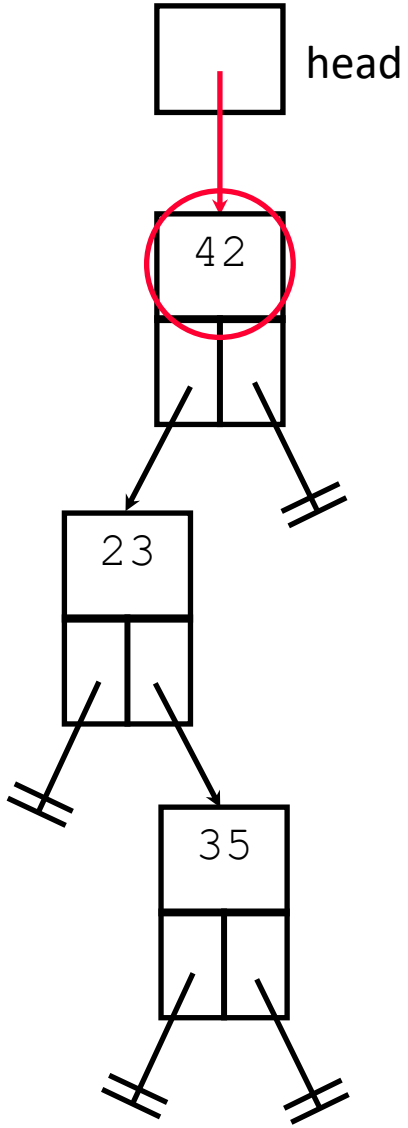
```
procedure Insert(
 cur iot in/out Ptr toa Node,
 data_in iot in num)
 if(cur = NIL) then
  cur <- new(Node)
  cur^.data <- data_in
  cur^.left <- NIL
  cur^.right <- NIL
 elseif(cur^.data > data_in)
  Insert(cur^.left, data_in)
 else
  Insert(cur^.right, data_in)
 endif
endprocedure // Insert
```



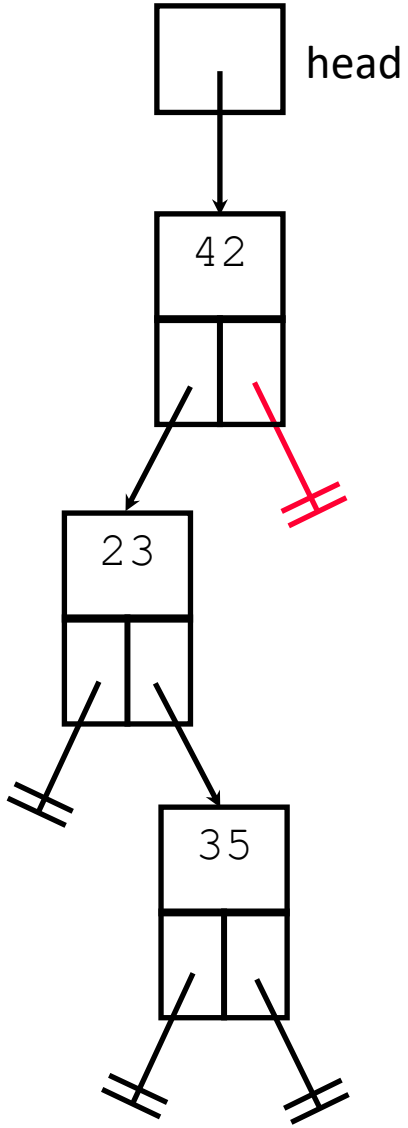
data_in = 47

```
procedure Insert(
 cur iot in/out Ptr toa Node,
 data_in iot in num)
 if(cur = NIL) then
  cur <- new(Node)
  cur^.data <- data_in
  cur^.left <- NIL
  cur^.right <- NIL
 elseif(cur^.data > data_in)
  Insert(cur^.left, data_in)
 else
  Insert(cur^.right, data_in)
 endif
endprocedure // Insert
```

```
data_in = 47
```



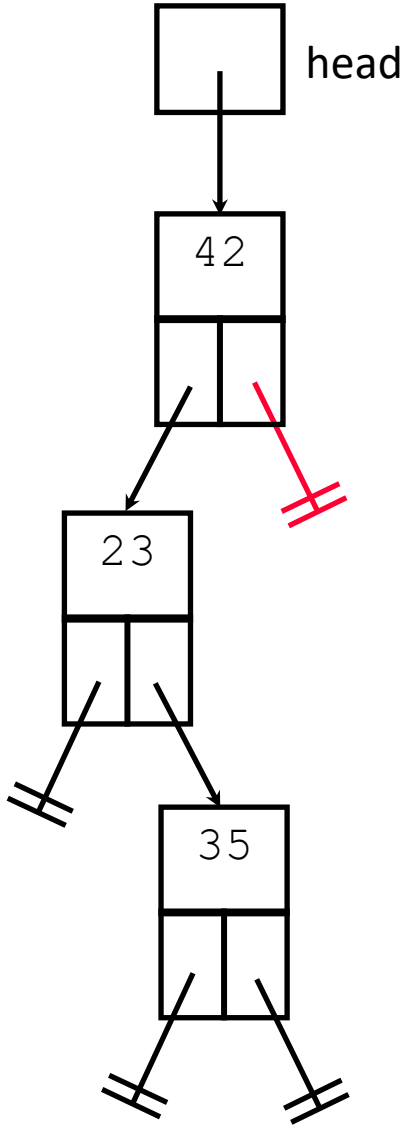
```
procedure Insert(
 cur iot in/out Ptr toa Node,
 data_in iot in num)
 if(cur = NIL) then
  cur <- new(Node)
  cur^.data <- data_in
  cur^.left <- NIL
  cur^.right <- NIL
 elseif(cur^.data > data_in)
  Insert(cur^.left, data_in)
 else
  Insert(cur^.right, data_in)
 endif
endprocedure // Insert
```



data_in = 47

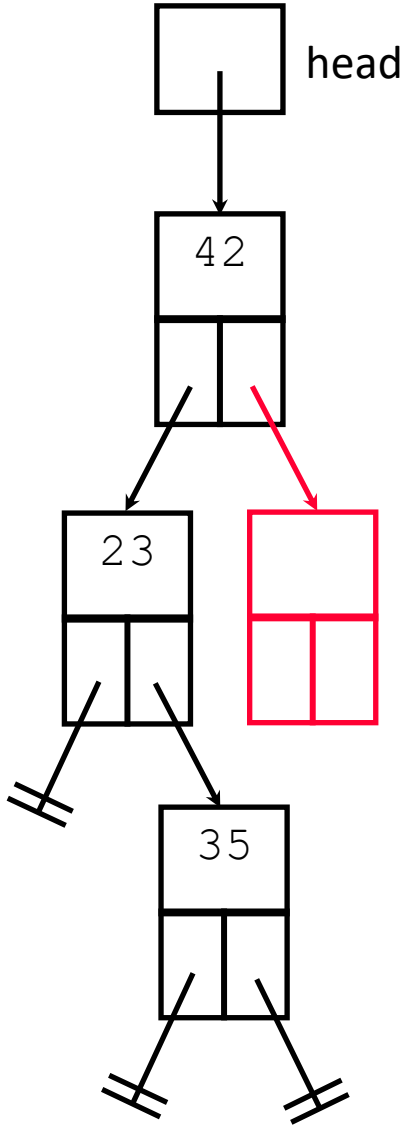
```
procedure Insert(
 cur iot in/out Ptr toa Node,
 data_in iot in num)
 if(cur = NIL) then
  cur <- new(Node)
  cur^.data <- data_in
  cur^.left <- NIL
  cur^.right <- NIL
 elseif(cur^.data > data_in)
  Insert(cur^.left, data_in)
 else
  Insert(cur^.right, data_in)
 endif
endprocedure // Insert
```

data_in = 47



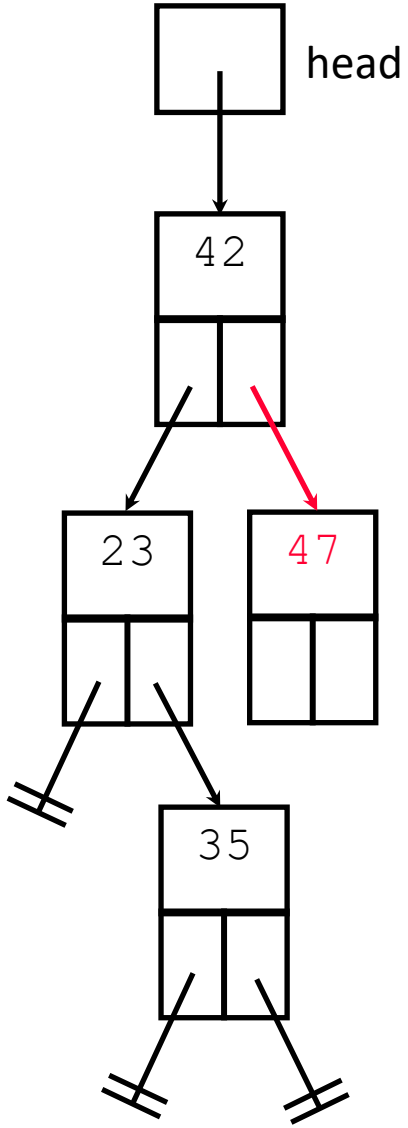
```
procedure Insert(
 cur iot in/out Ptr toa Node,
 data_in iot in num)
 if(cur = NIL) then
  cur <- new(Node)
  cur^.data <- data_in
  cur^.left <- NIL
  cur^.right <- NIL
 elseif(cur^.data > data_in)
  Insert(cur^.left, data_in)
 else
  Insert(cur^.right, data_in)
 endif
endprocedure // Insert
```

```
data_in = 47
```



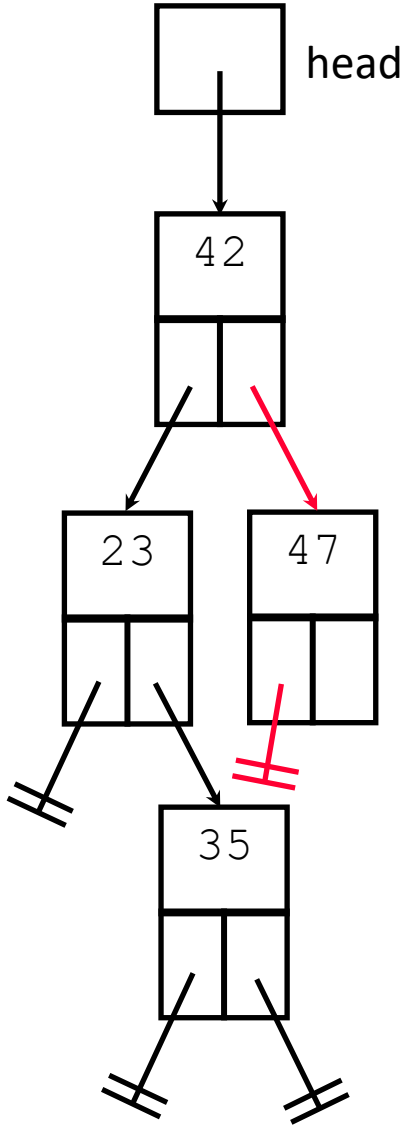
```
procedure Insert(
 cur iot in/out Ptr toa Node,
 data_in iot in num)
 if(cur = NIL) then
  cur <- new(Node)
  cur^.data <- data_in
  cur^.left <- NIL
  cur^.right <- NIL
 elseif(cur^.data > data_in)
  Insert(cur^.left, data_in)
 else
  Insert(cur^.right, data_in)
 endif
endprocedure // Insert
```

data_in = 47



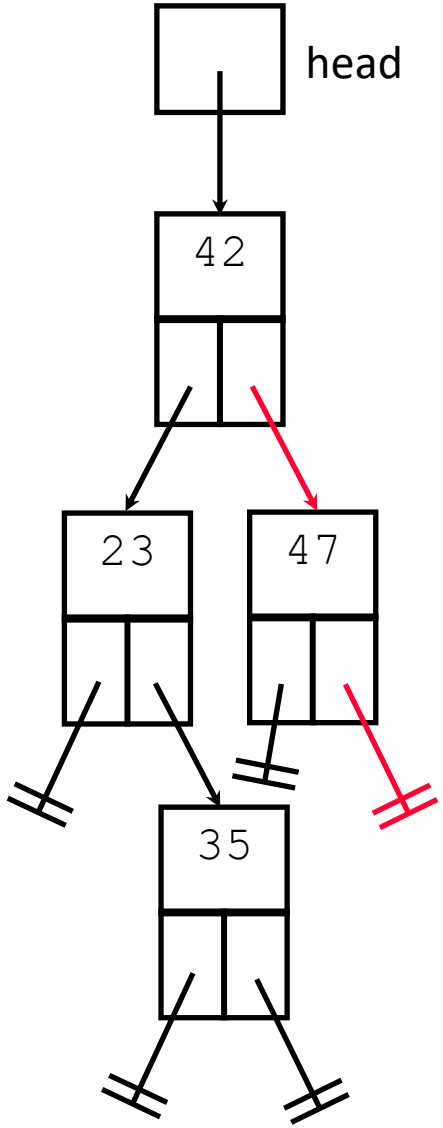
```
procedure Insert(
 cur iot in/out Ptr toa Node,
 data_in iot in num)
 if(cur = NIL) then
  cur <- new(Node)
  cur^.data <- data_in
  cur^.left <- NIL
  cur^.right <- NIL
 elseif(cur^.data > data_in)
  Insert(cur^.left, data_in)
 else
  Insert(cur^.right, data_in)
 endif
endprocedure // Insert
```

```
data_in = 47
```



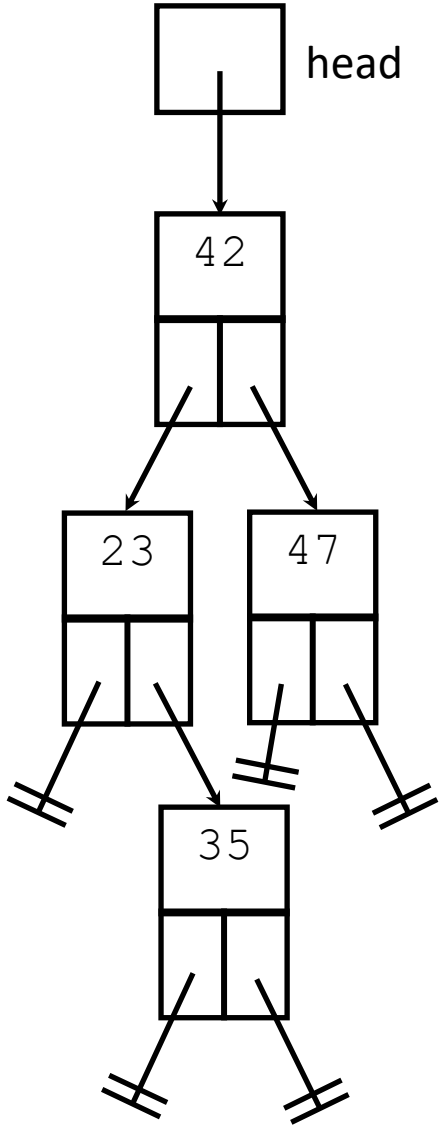
```
procedure Insert(
 cur iot in/out Ptr toa Node,
 data_in iot in num)
 if(cur = NIL) then
  cur <- new(Node)
  cur^.data <- data_in
  cur^.left <- NIL
  cur^.right <- NIL
 elseif(cur^.data > data_in)
  Insert(cur^.left, data_in)
 else
  Insert(cur^.right, data_in)
 endif
endprocedure // Insert
```

```
data_in = 47
```

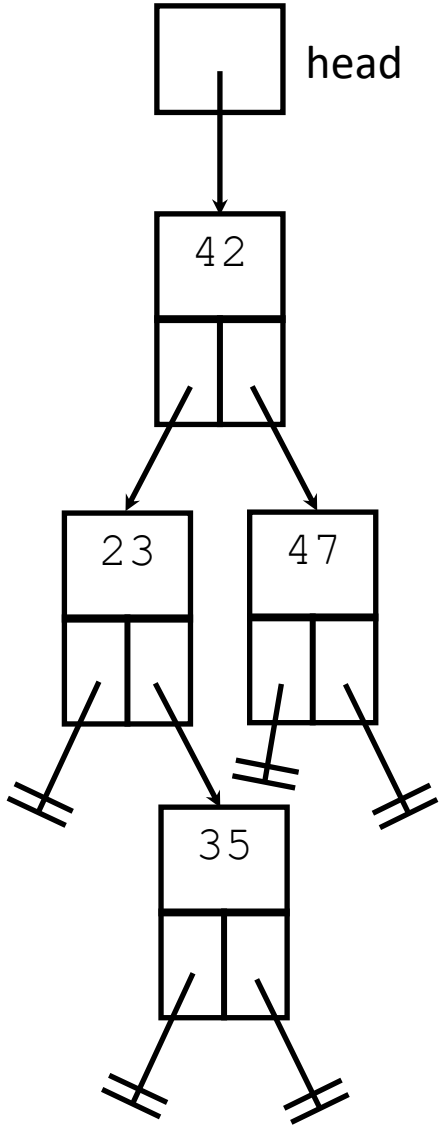


```
procedure Insert(
 cur iot in/out Ptr toa Node,
 data_in iot in num)
 if(cur = NIL) then
  cur <- new(Node)
  cur^.data <- data_in
  cur^.left <- NIL
  cur^.right <- NIL
 elseif(cur^.data > data_in)
  Insert(cur^.left, data_in)
 else
  Insert(cur^.right, data_in)
 endif
endprocedure // Insert
```

```
data_in = 47
```

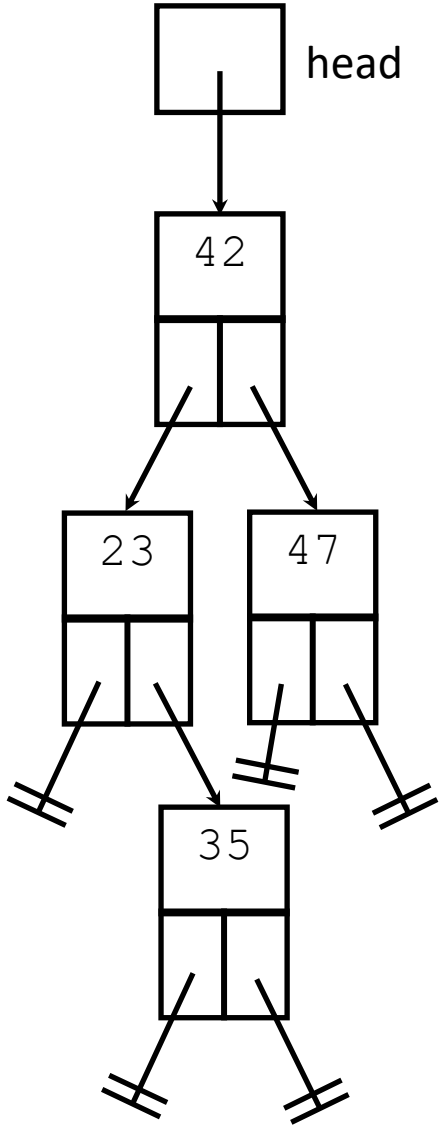


```
procedure Insert(
 cur iot in/out Ptr toa Node,
 data_in iot in num)
 if(cur = NIL) then
  cur <- new(Node)
  cur^.data <- data_in
  cur^.left <- NIL
  cur^.right <- NIL
 elseif(cur^.data > data_in)
  Insert(cur^.left, data_in)
 else
  Insert(cur^.right, data_in)
 endif
endprocedure // Insert
```



data_in = 47

```
.
.
Insert(head, 23)
Insert(head, 35)
Insert(head, 47)
.
.
```



Summary

- Preserve “search” structure!
- Inserting involves 2 steps:
  - Find the correct location
    - For a BST insert, always insert at the “bottom” of the tree
  - Do commands to add node
    - Create node
    - Add data
    - Make left and right pointers point to nil

DELETION OF A NODE

Deleting a Node in BST

- **We have a Binary Search Tree and want to remove some element based upon a match.**

- **Must preserve “search” property**
- **Must not lose any elements (i.e. only remove the one element)**

BST Deletion

- **Search for desired item.**
- **If not found, then return NIL or print error.**
- **If found, perform steps necessary to accomplish removal from the tree.**

Four Cases for Deletion

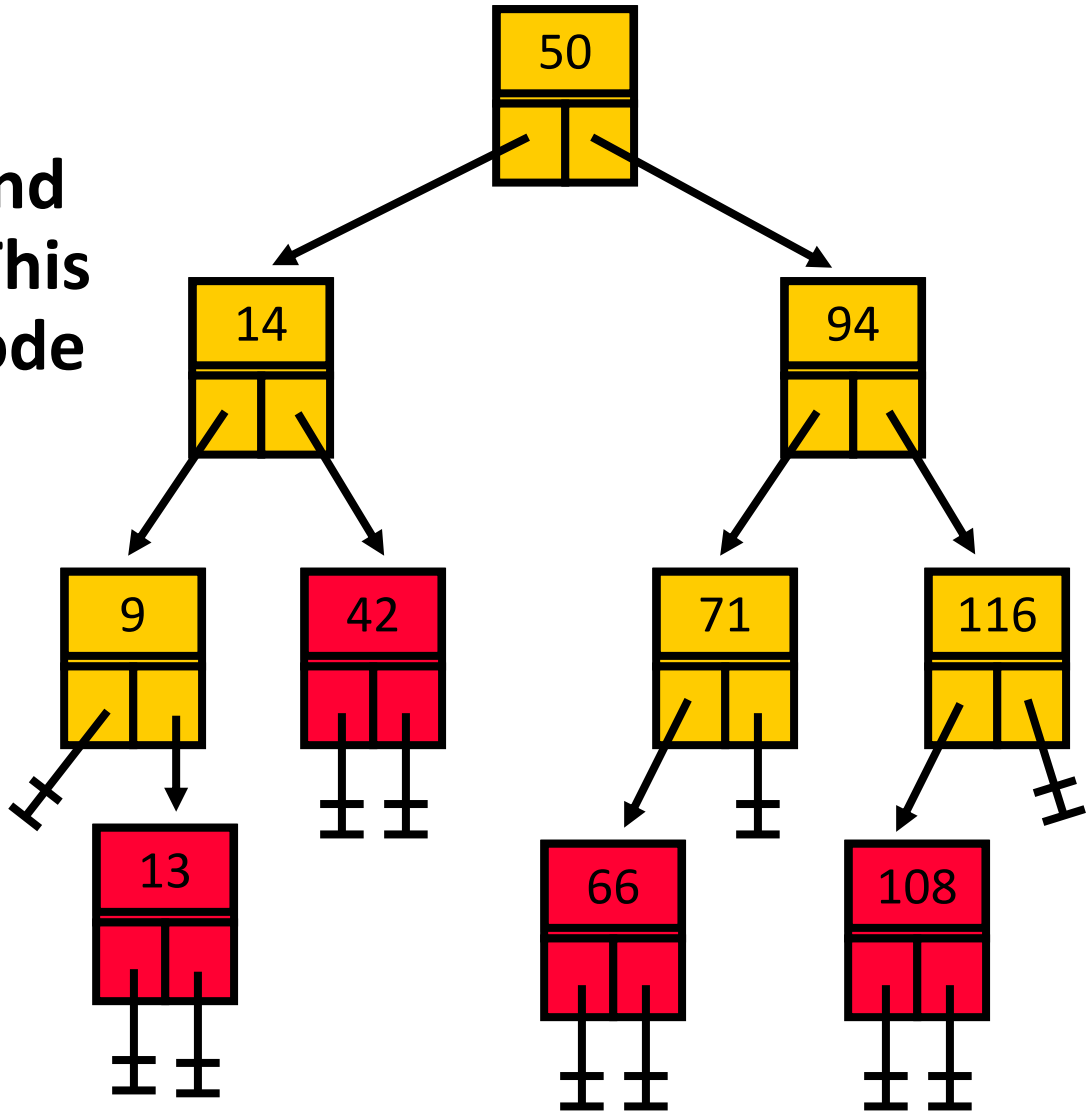
- **Delete a leaf node**
- **Delete a node with only one child (left)**
- **Delete a node with only one child (right)**
- **Delete a node with two children**

Cases 2 and 3 are comparable and only need slight changes in the conditional statement used

Delete a Leaf Node

Simply use an "in/out" pointer and assign it to "nil". This will remove the node from the tree.

```
cur <- nil
```

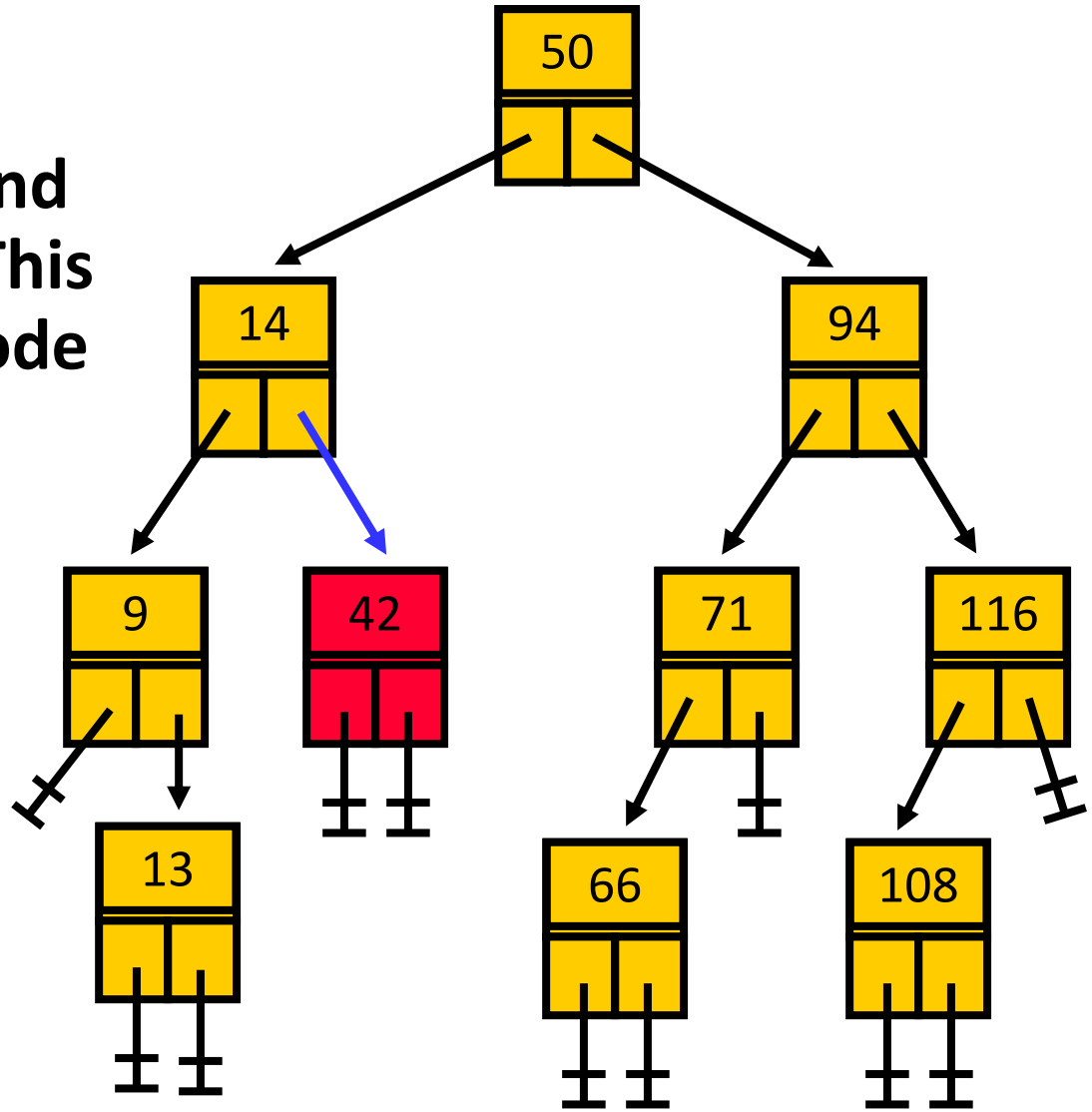


Delete a Leaf Node

Simply use an "in/out" pointer and assign it to "nil". This will remove the node from the tree.

```
cur <- nil
```

Let's delete 42.

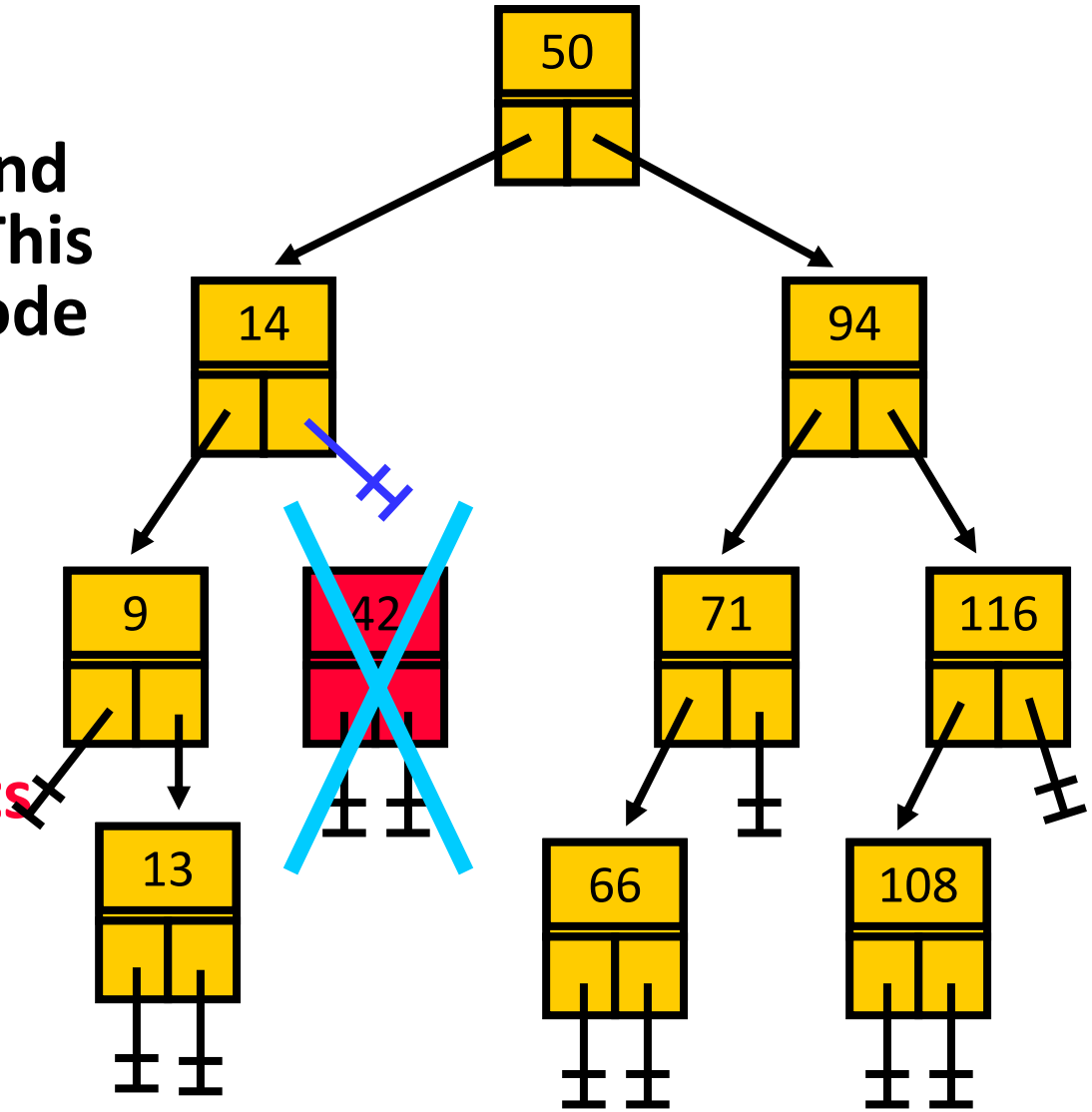


Delete a Leaf Node

Simply use an "in/out" pointer and assign it to "nil". This will remove the node from the tree.

```
cur <- nil
```

Move the pointer; now nothing points to the node.

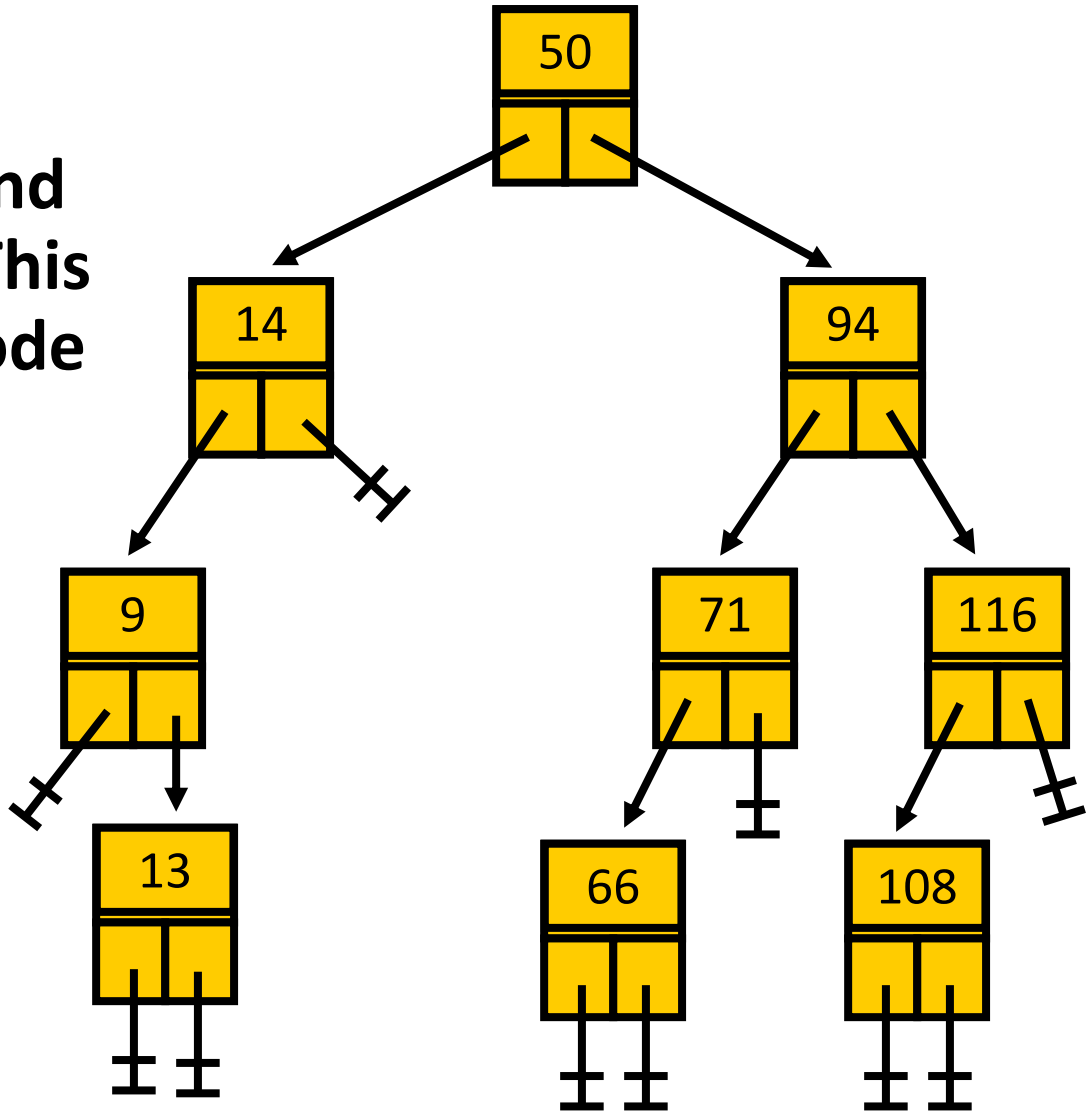


Delete a Leaf Node

Simply use an "in/out" pointer and assign it to "nil". This will remove the node from the tree.

```
cur <- nil
```

The resulting tree.



Delete a Node with One Child

Use an "in/out" pointer.

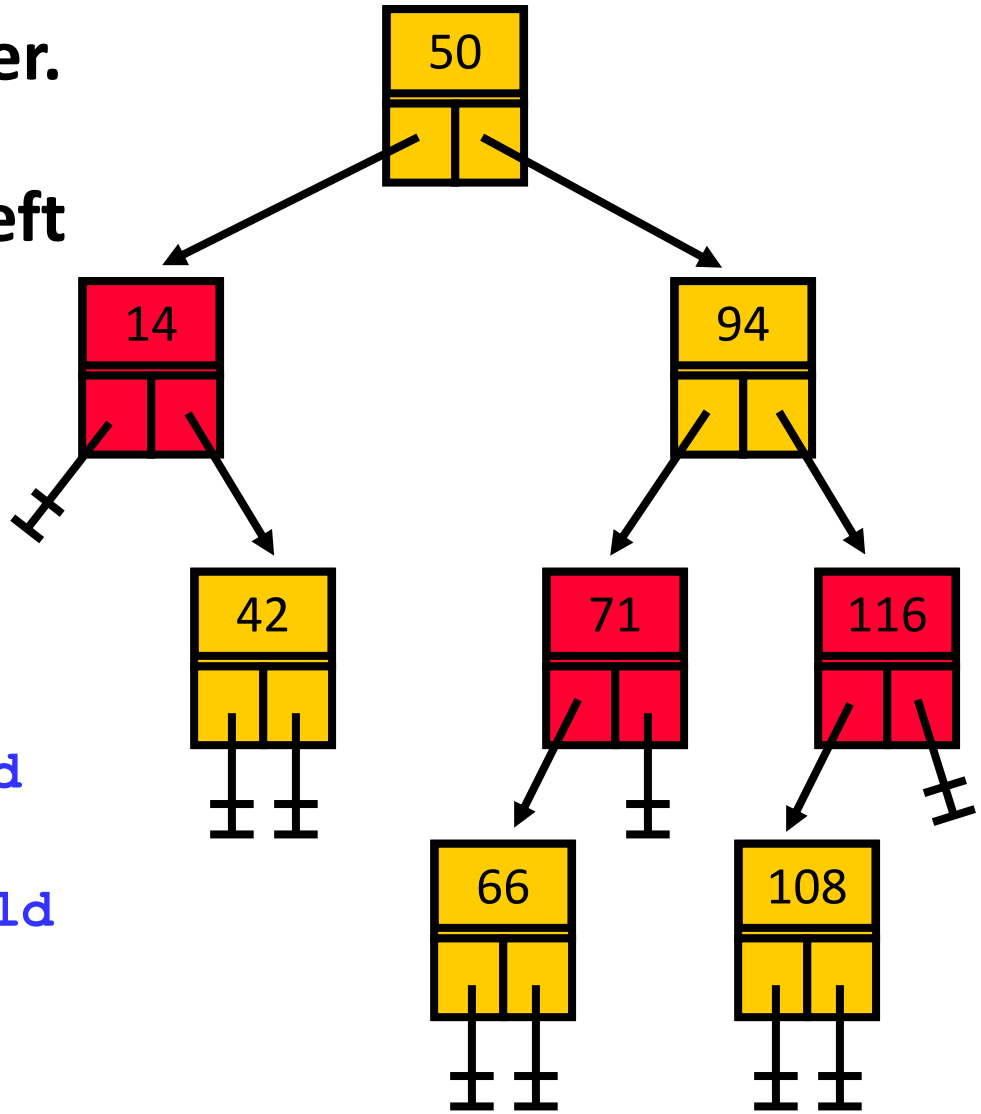
Determine if it has a left or a right child.

Point the current pointer to the appropriate child:

```
cur <- cur^.left_child
```

or

```
cur <- cur^.right_child
```



Delete a Node with One Child

Use an "in/out" pointer.

Determine if it has a left or a right child.

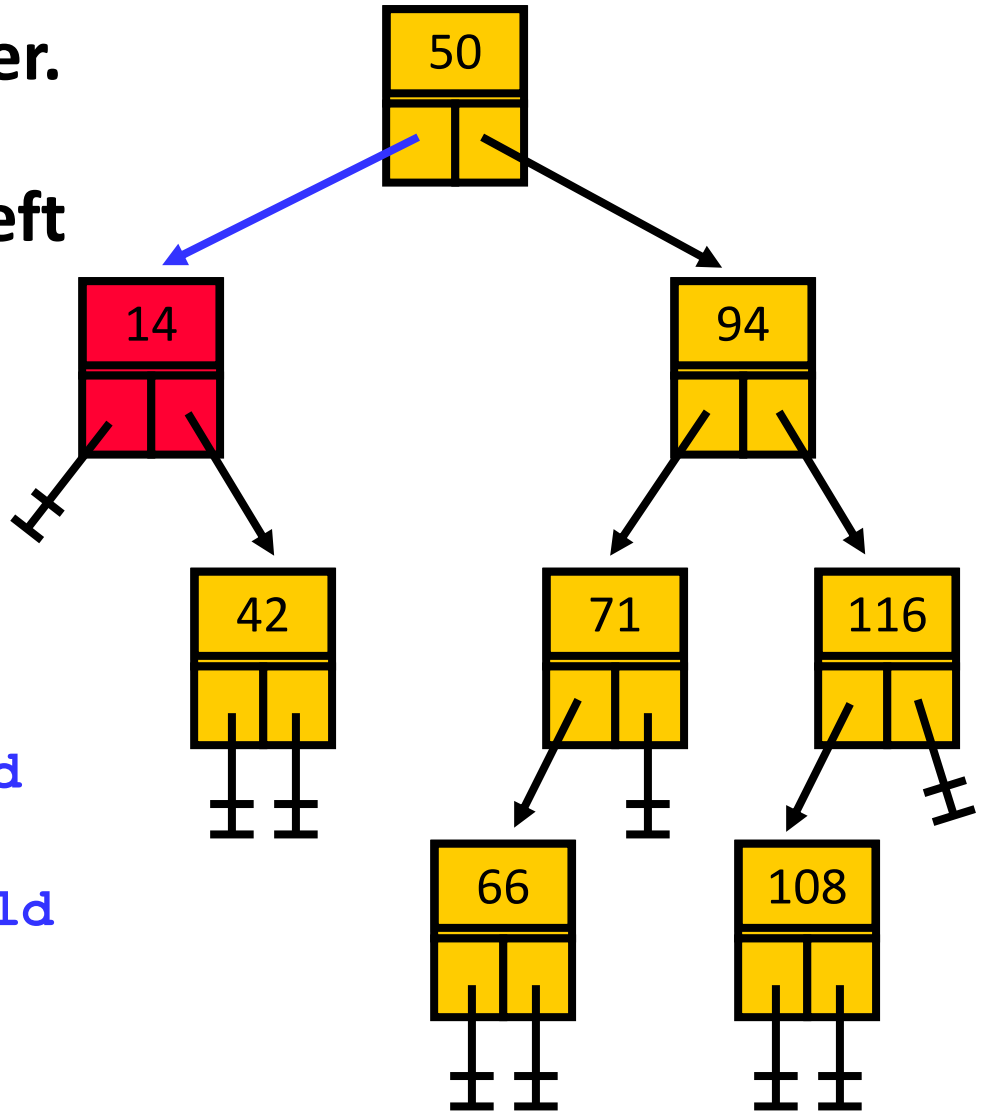
Point the current pointer to the appropriate child:

```
cur <- cur^.left_child
```

or

```
cur <- cur^.right_child
```

Let's delete 14.



Delete a Node with One Child

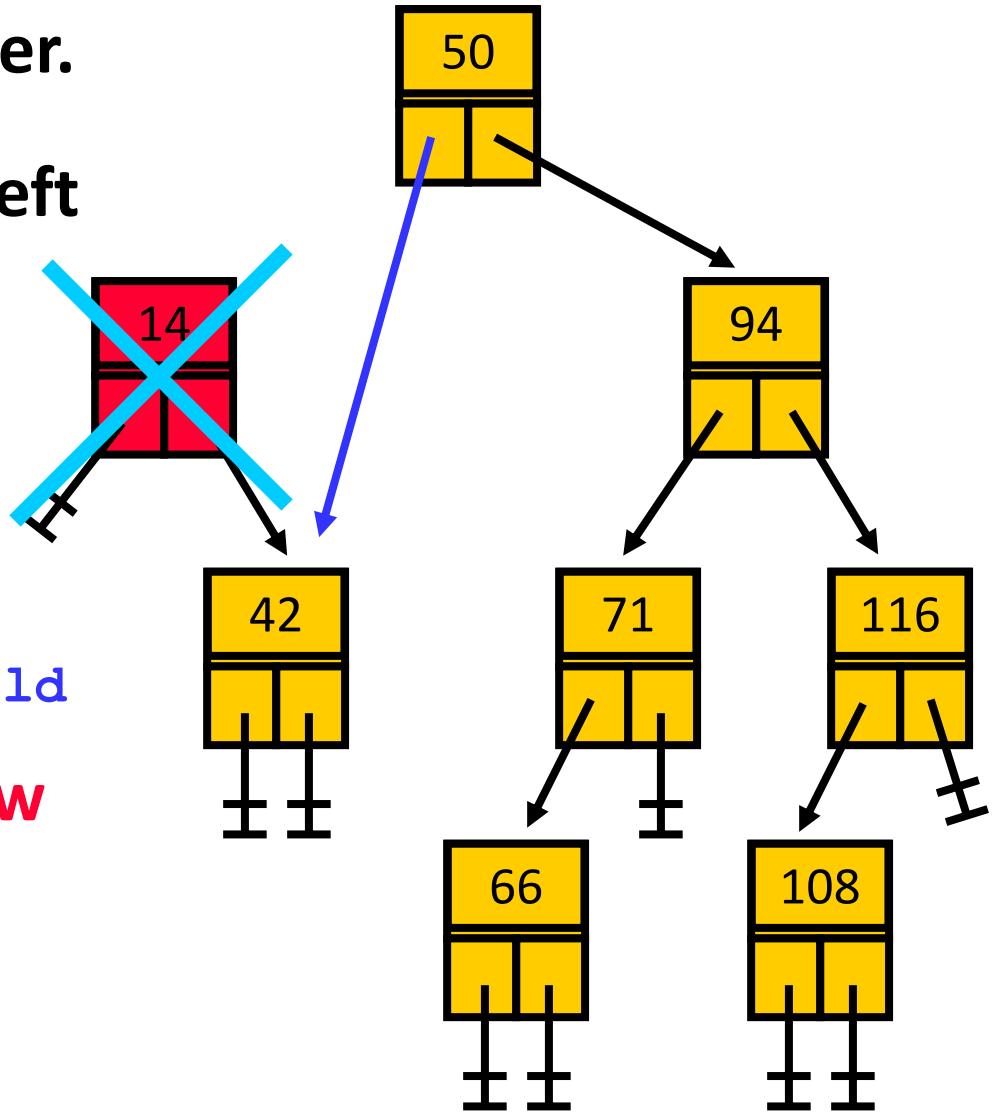
Use an "in/out" pointer.

Determine if it has a left or a right child.

Point the current pointer to the appropriate child:

```
cur <- cur^.right_child
```

Move the pointer; now nothing points to the node.



Delete a Node with One Child

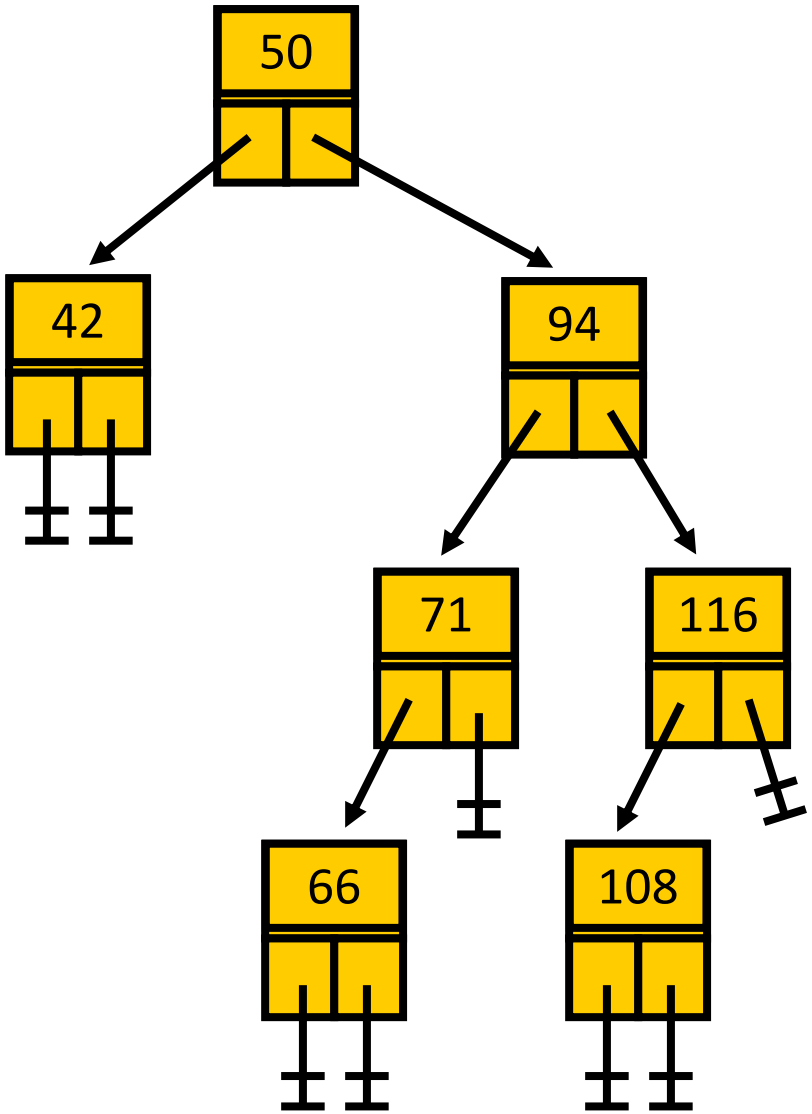
Use an "in/out" pointer.

Determine if it has a left or a right child.

Point the current pointer to the appropriate child:

```
cur <- cur^.right_child
```

The resulting tree.



Delete a Node with One Child

Use an "in/out" pointer.

Determine if it has a left or a right child.

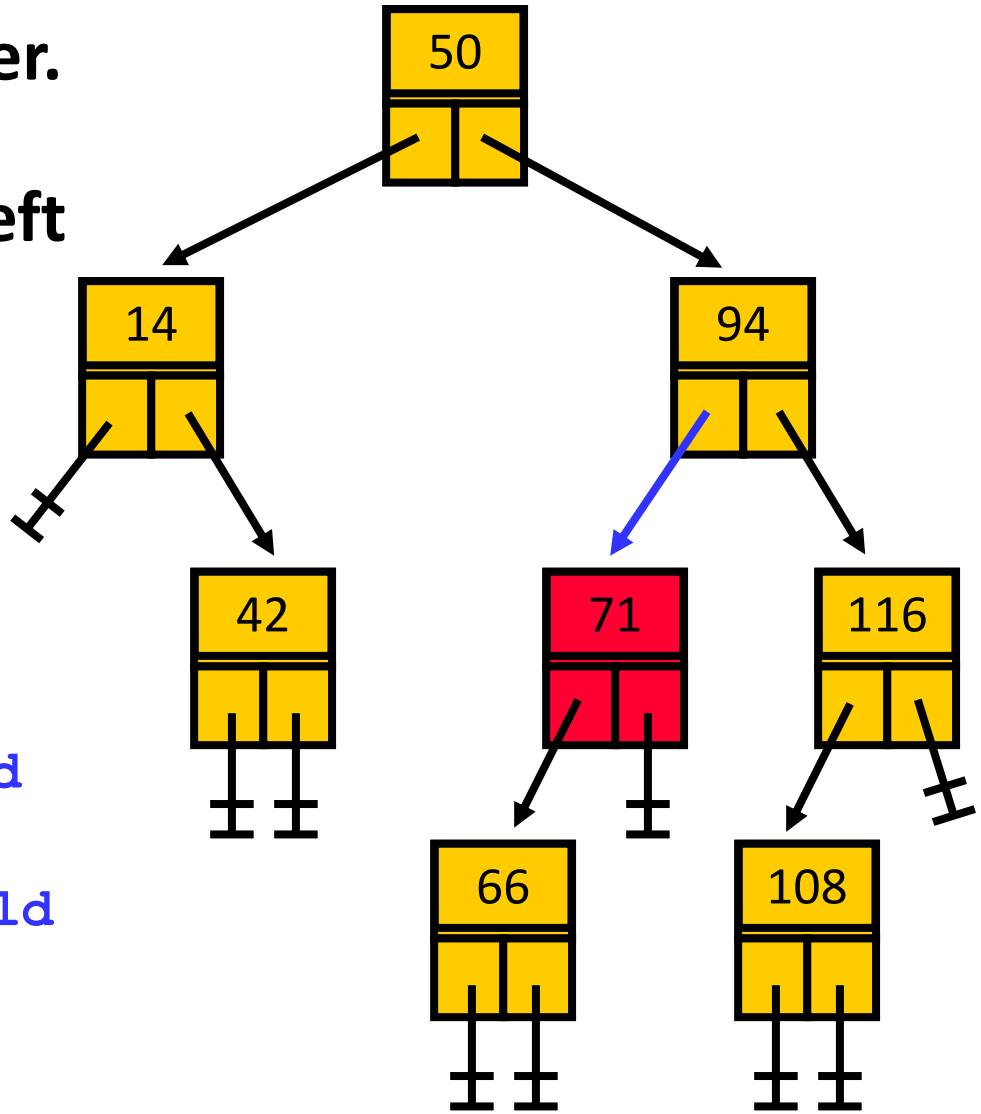
Point the current pointer to the appropriate child:

```
cur <- cur^.left_child
```

or

```
cur <- cur^.right_child
```

Let's delete 71.



Delete a Node with One Child

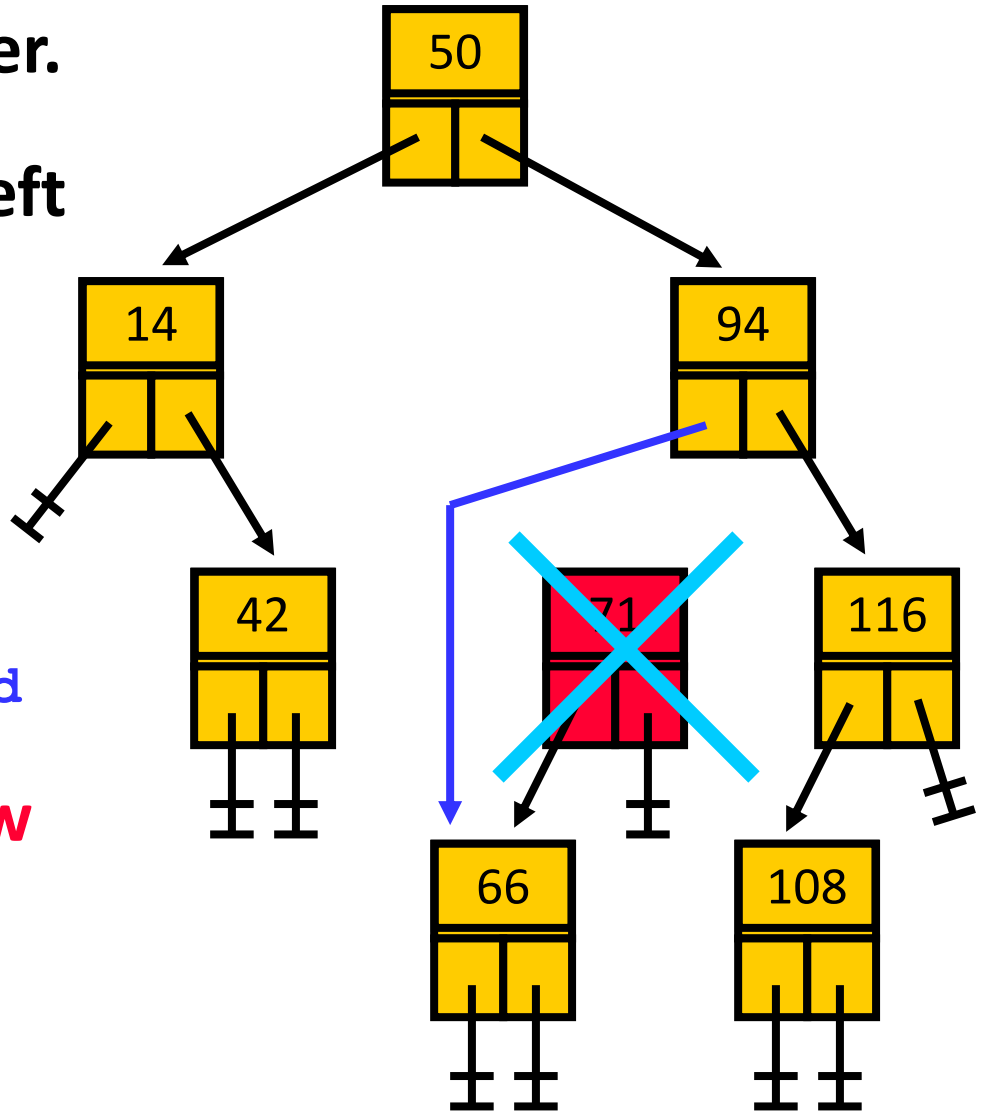
Use an "in/out" pointer.

Determine if it has a left or a right child.

Point the current pointer to the appropriate child:

```
cur <- cur^.left_child
```

Move the pointer; now nothing points to the node.



Delete a Node with One Child

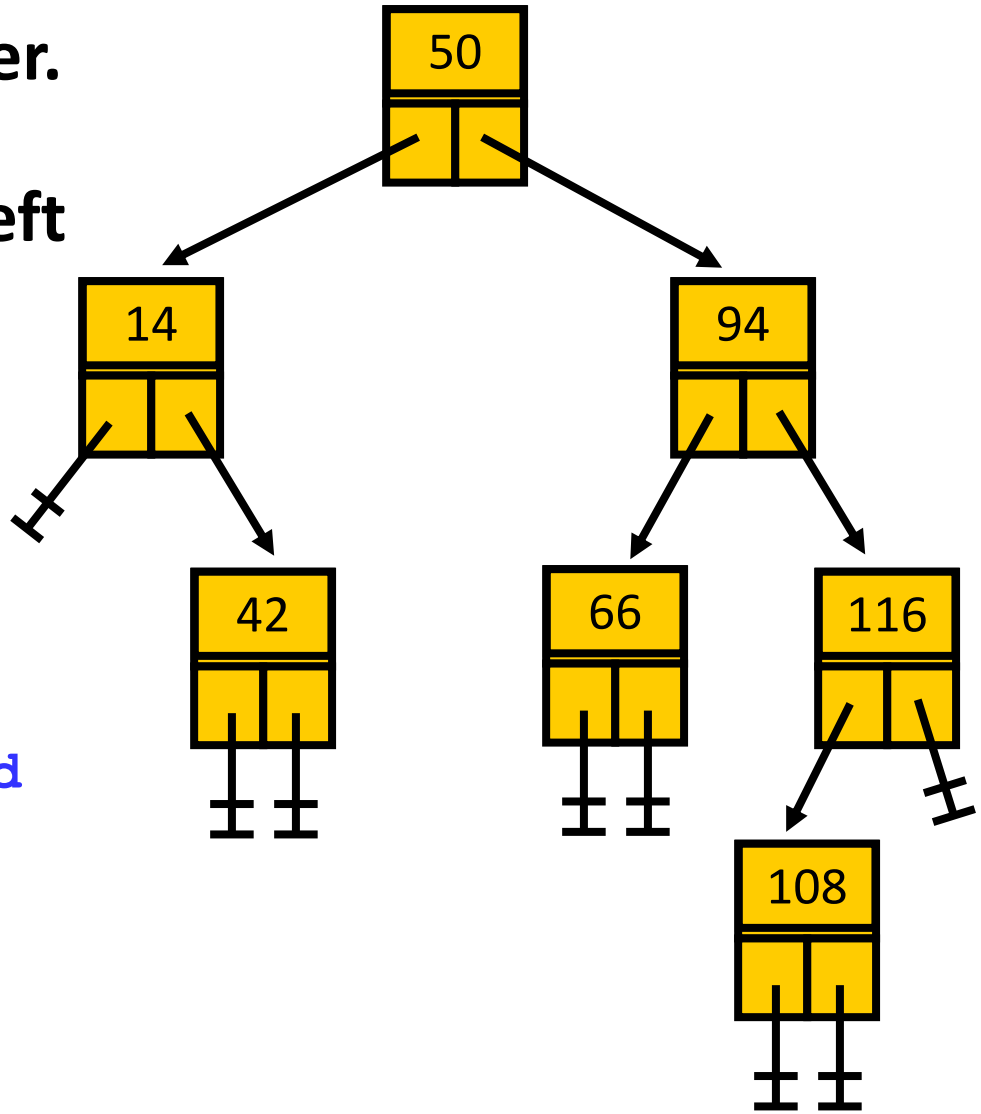
Use an "in/out" pointer.

Determine if it has a left or a right child.

Point the current pointer to the appropriate child:

```
cur <- cur^.left_child
```

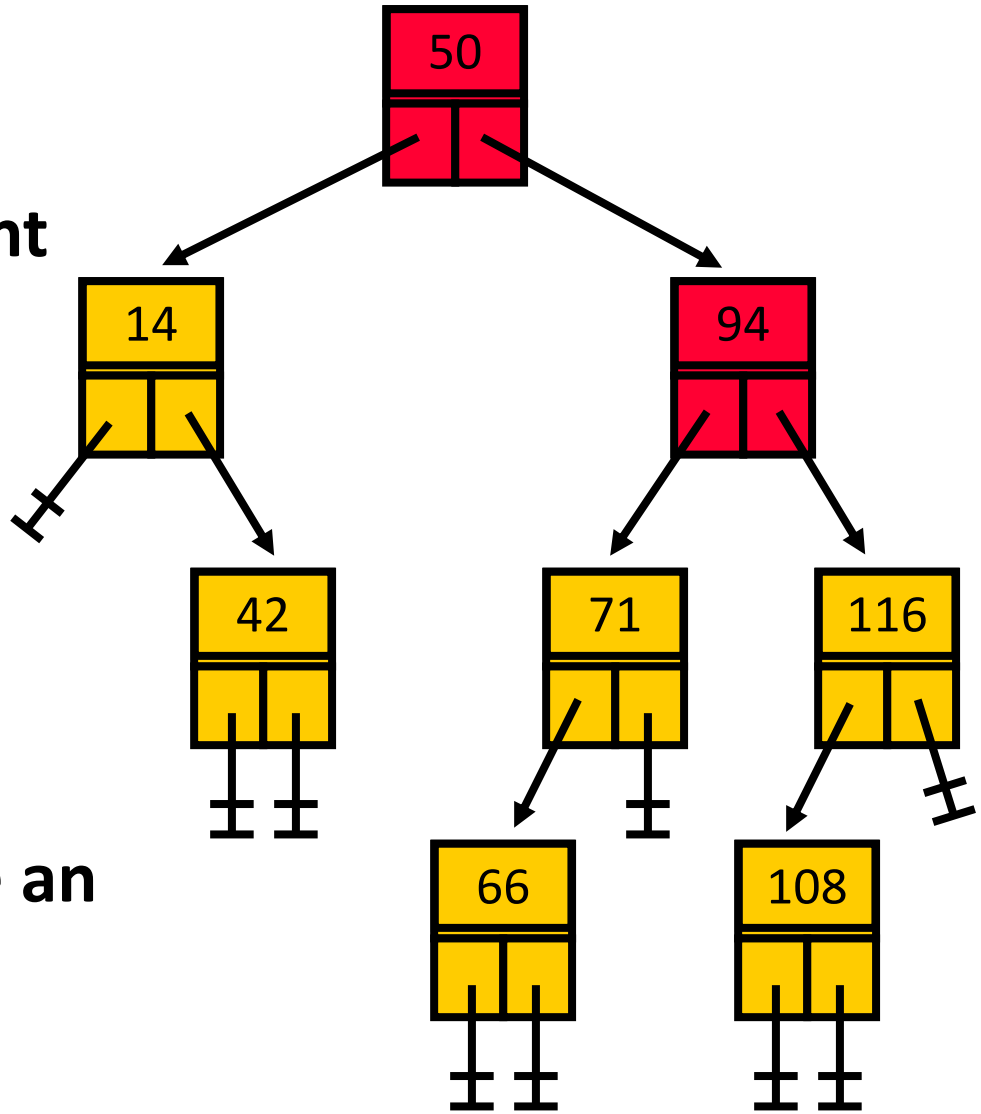
The resulting tree.



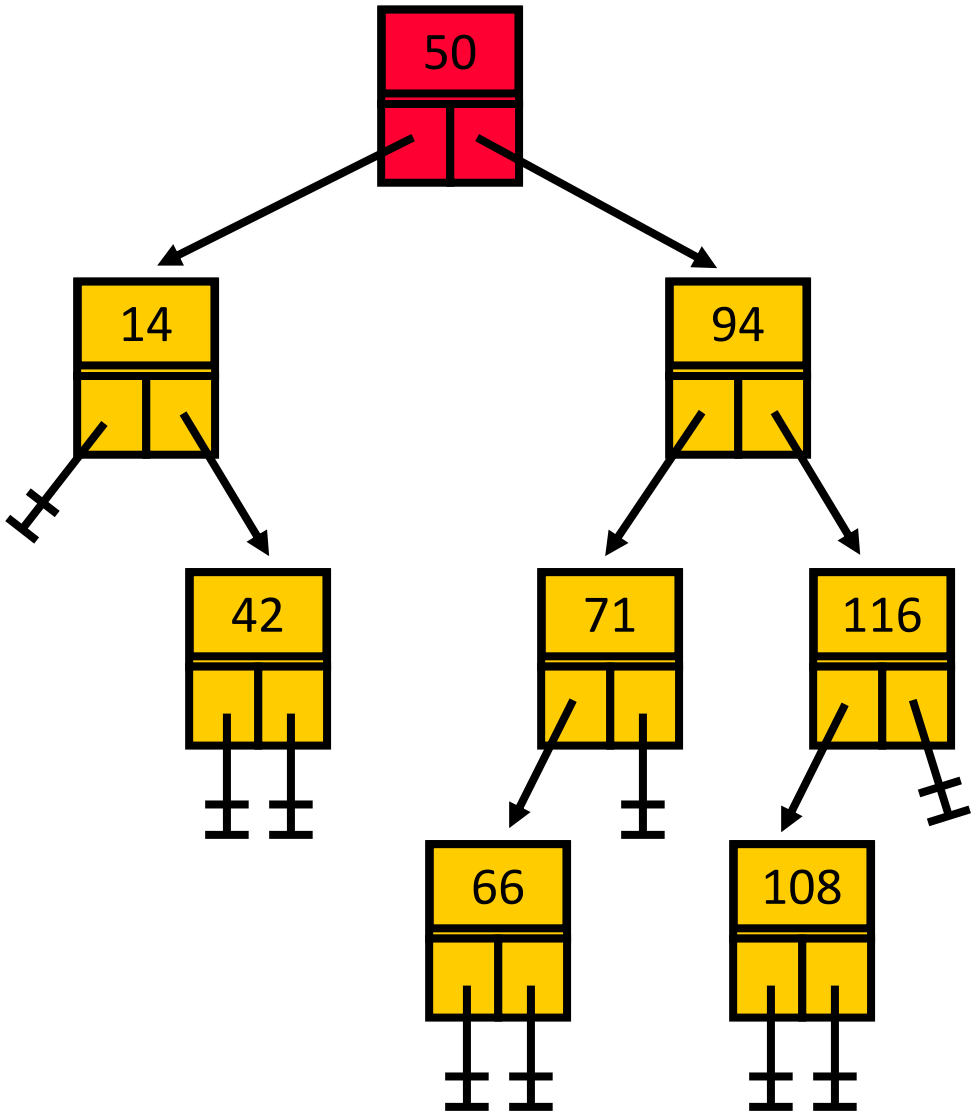
Delete a Node with Two Children

Copy a replacement value from a descendant node.
- **Largest from left**
- **Smallest from right**

Then delete that descendant node to remove the duplicate value.
- **We know this will be an easier case.**

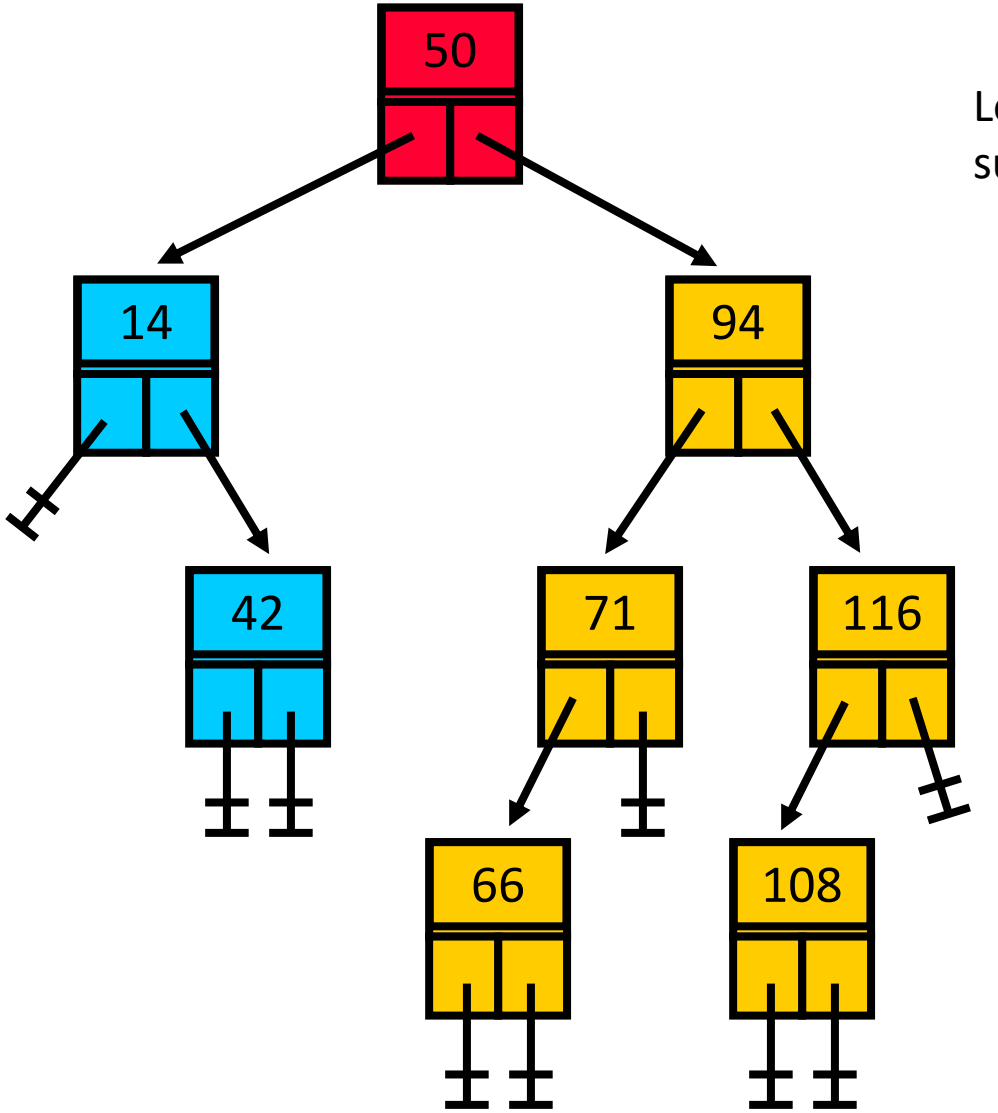


Delete a Node with Two Children



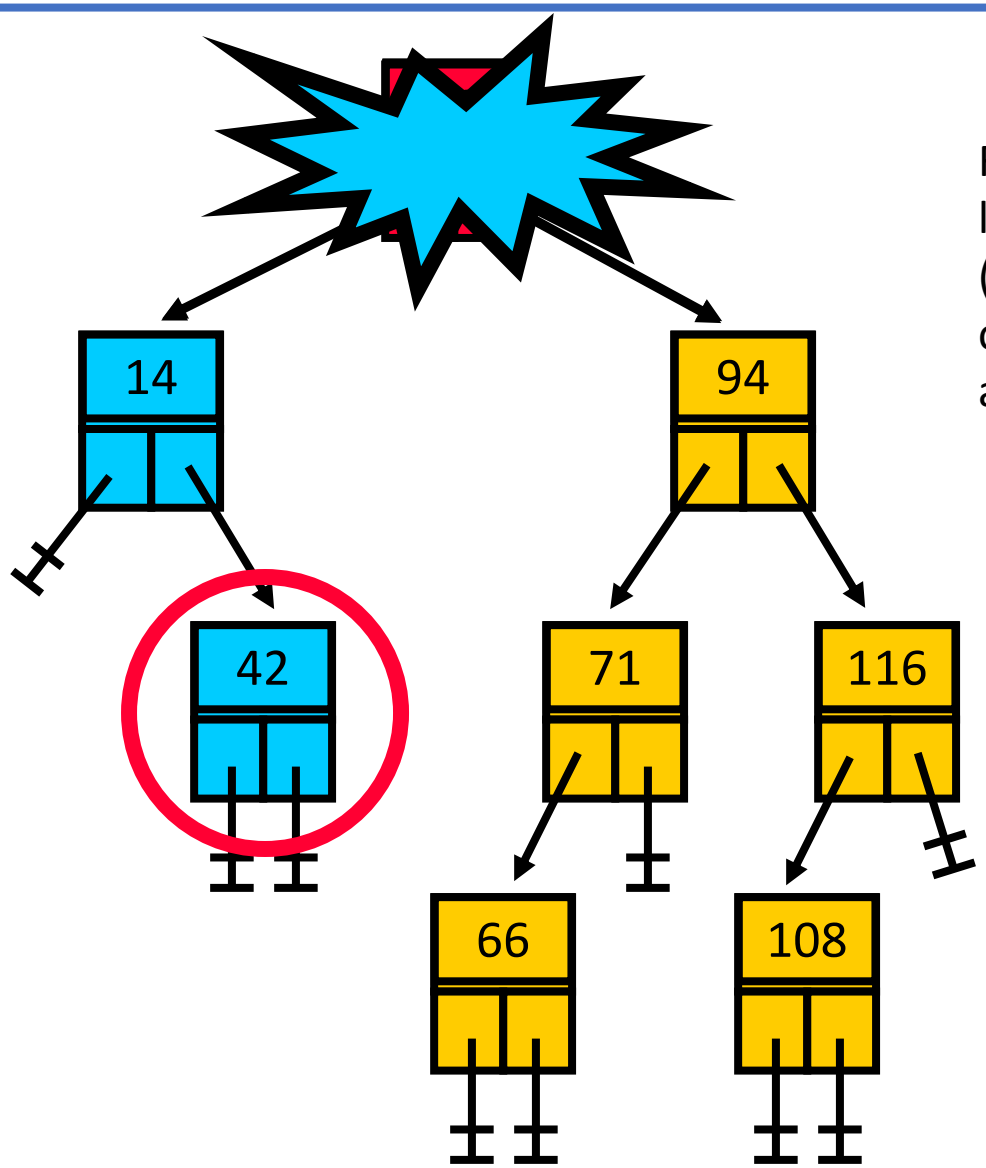
Let's delete 50.

Delete a Node with Two Children



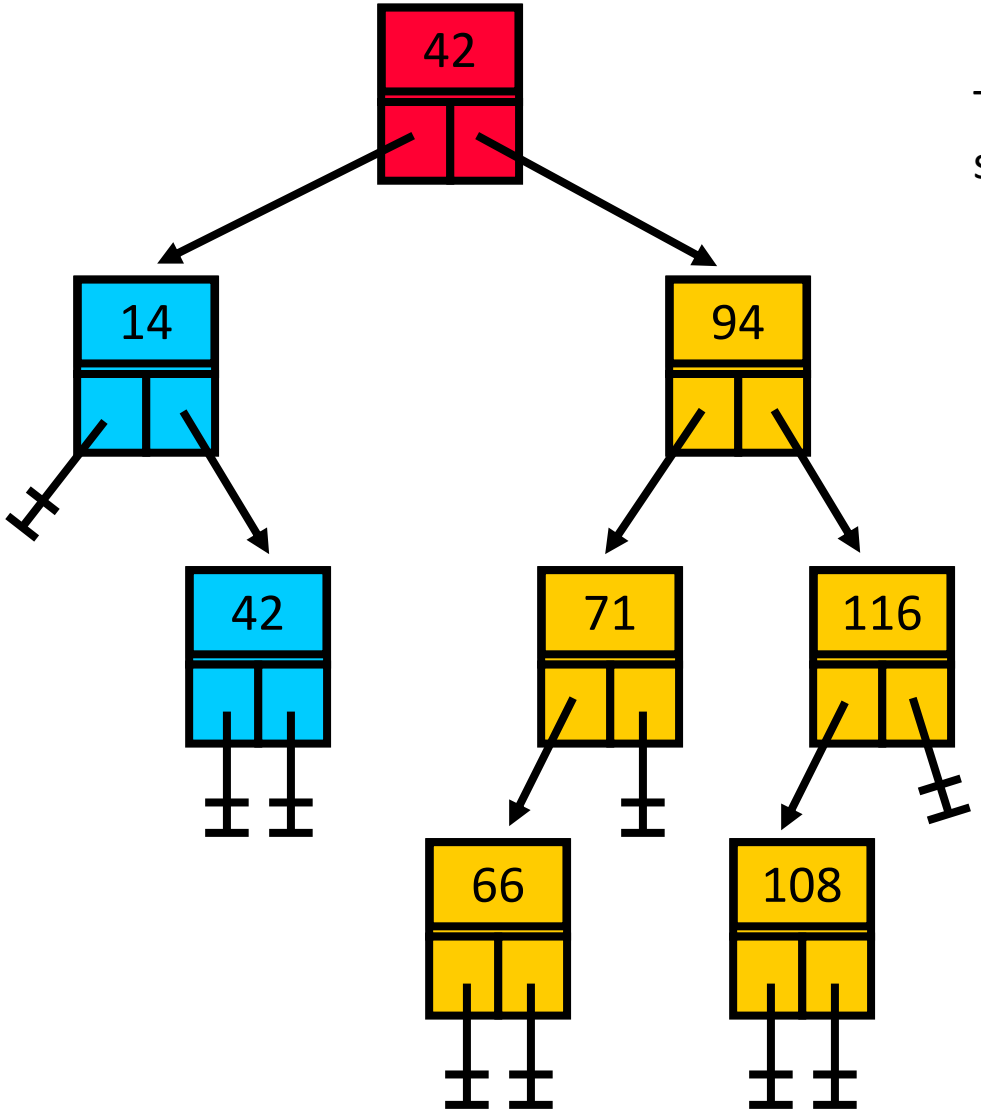
Look to the left sub-tree.

Delete a Node with Two Children



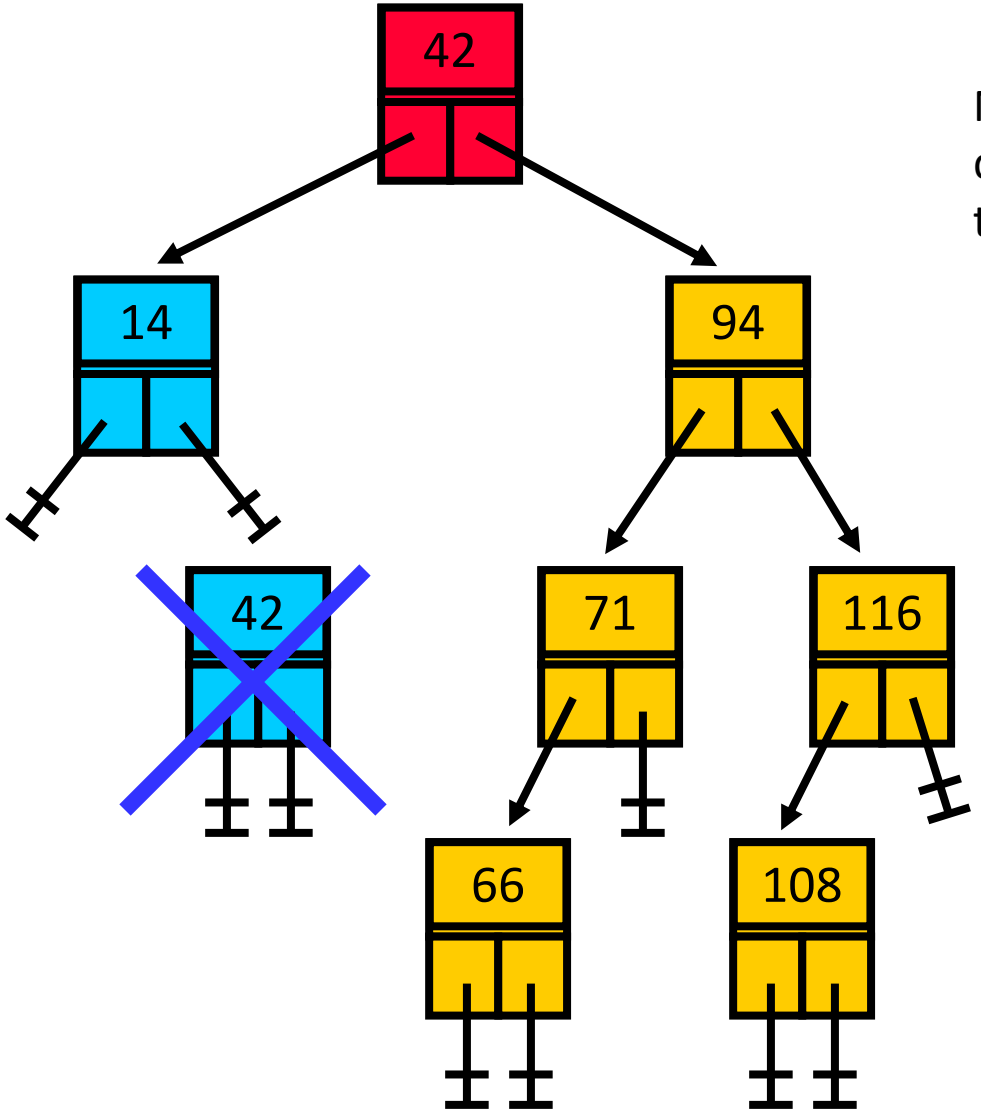
Find and copy the largest value (this will erase the old value but creates a duplicate).

Delete a Node with Two Children



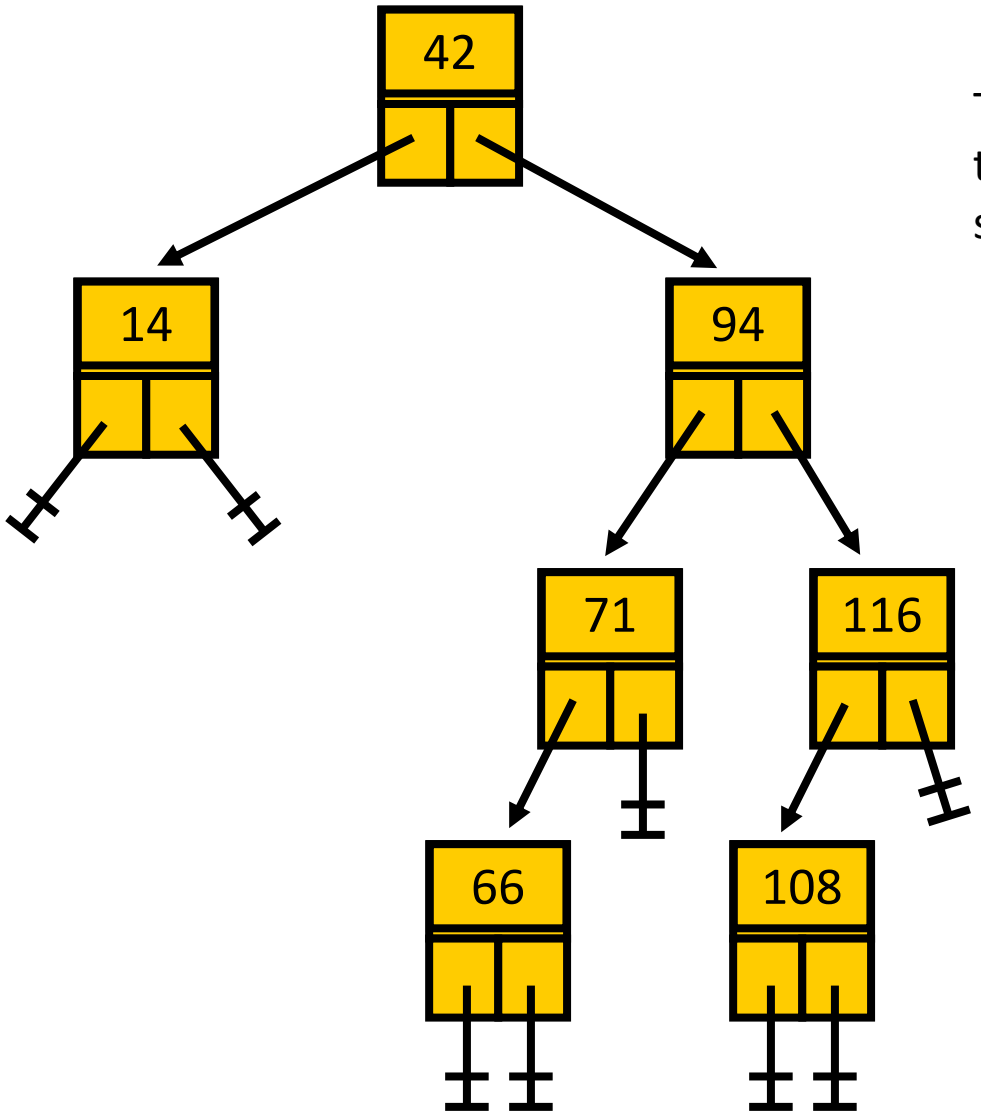
The resulting tree so far.

Delete a Node with Two Children



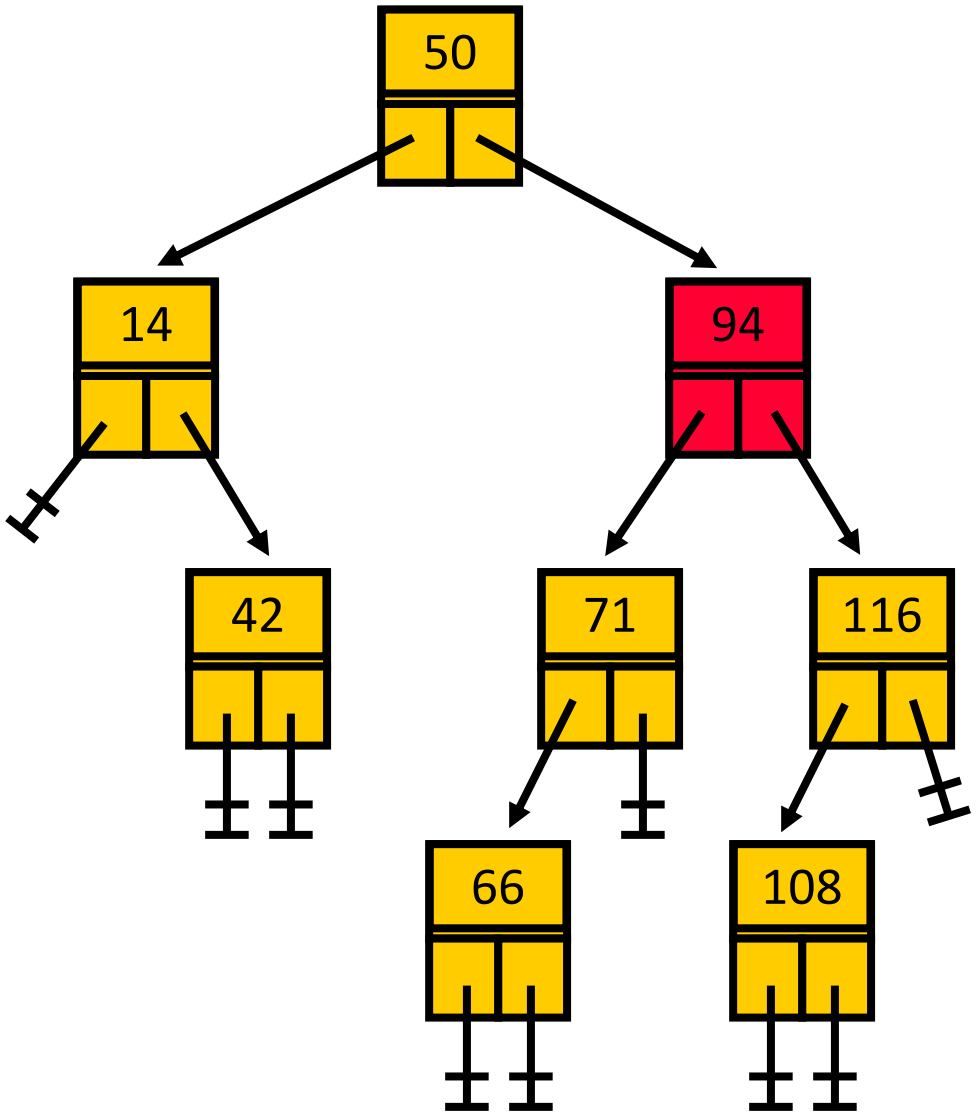
Now delete the duplicate from the left sub-tree.

Delete a Node with Two Children



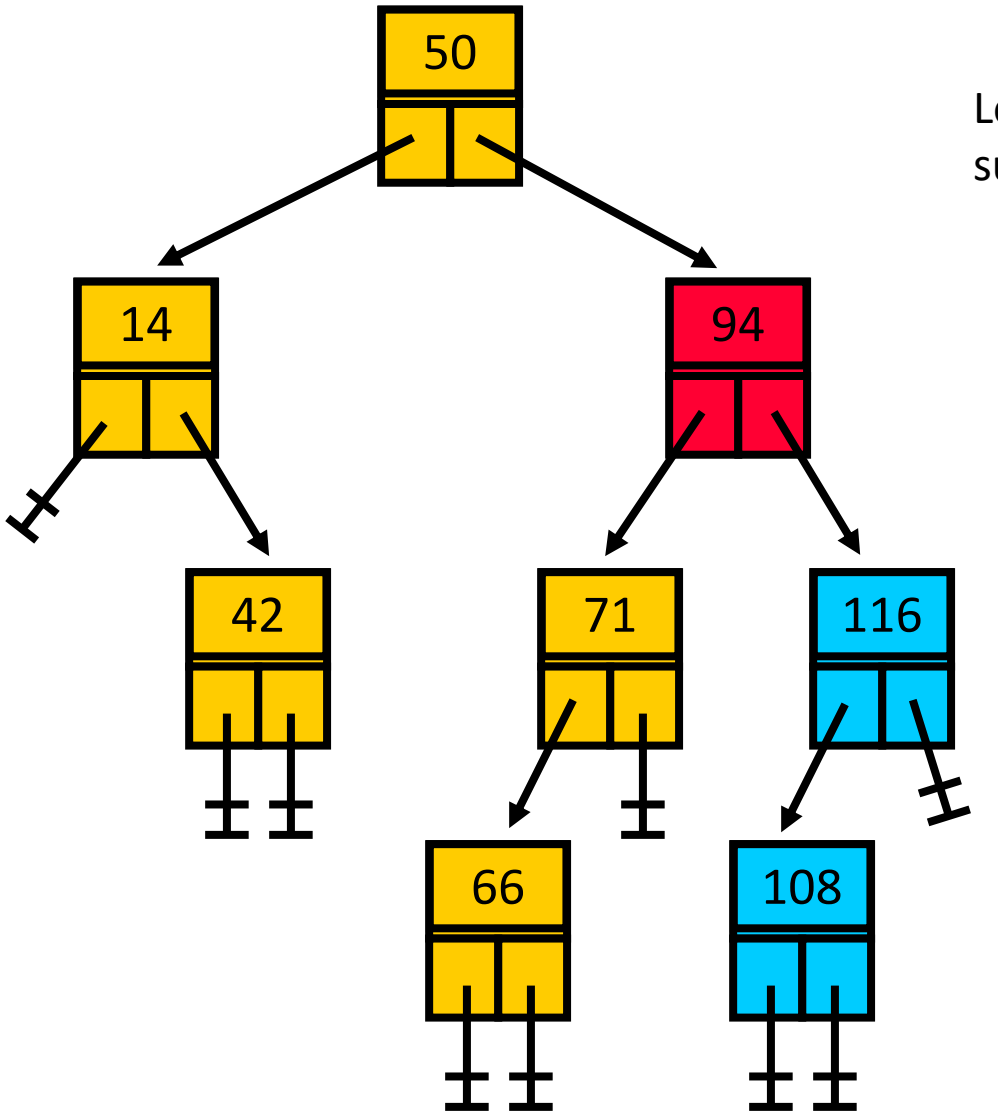
The final resulting tree – still has search structure.

Delete a Node with Two Children



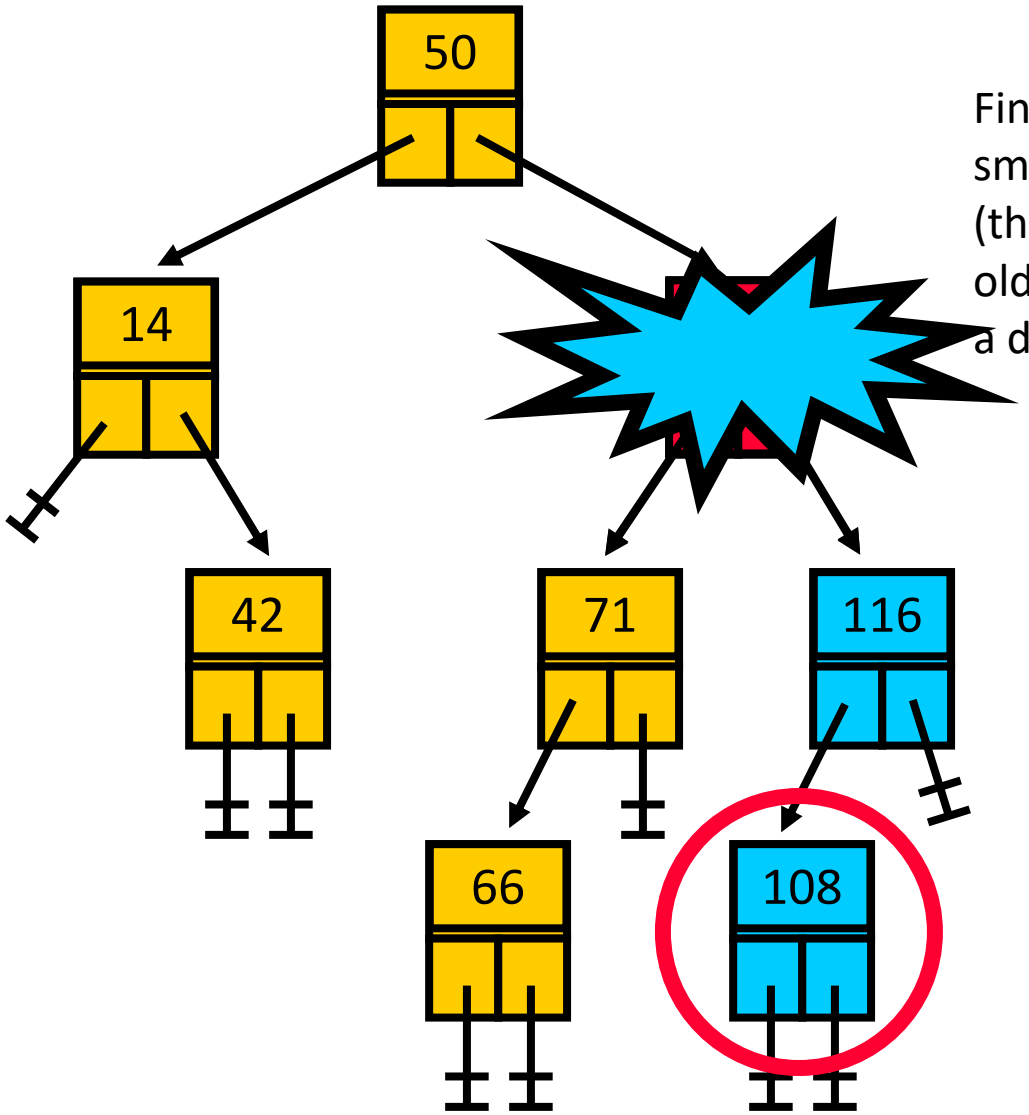
Let's delete 94.

Delete a Node with Two Children



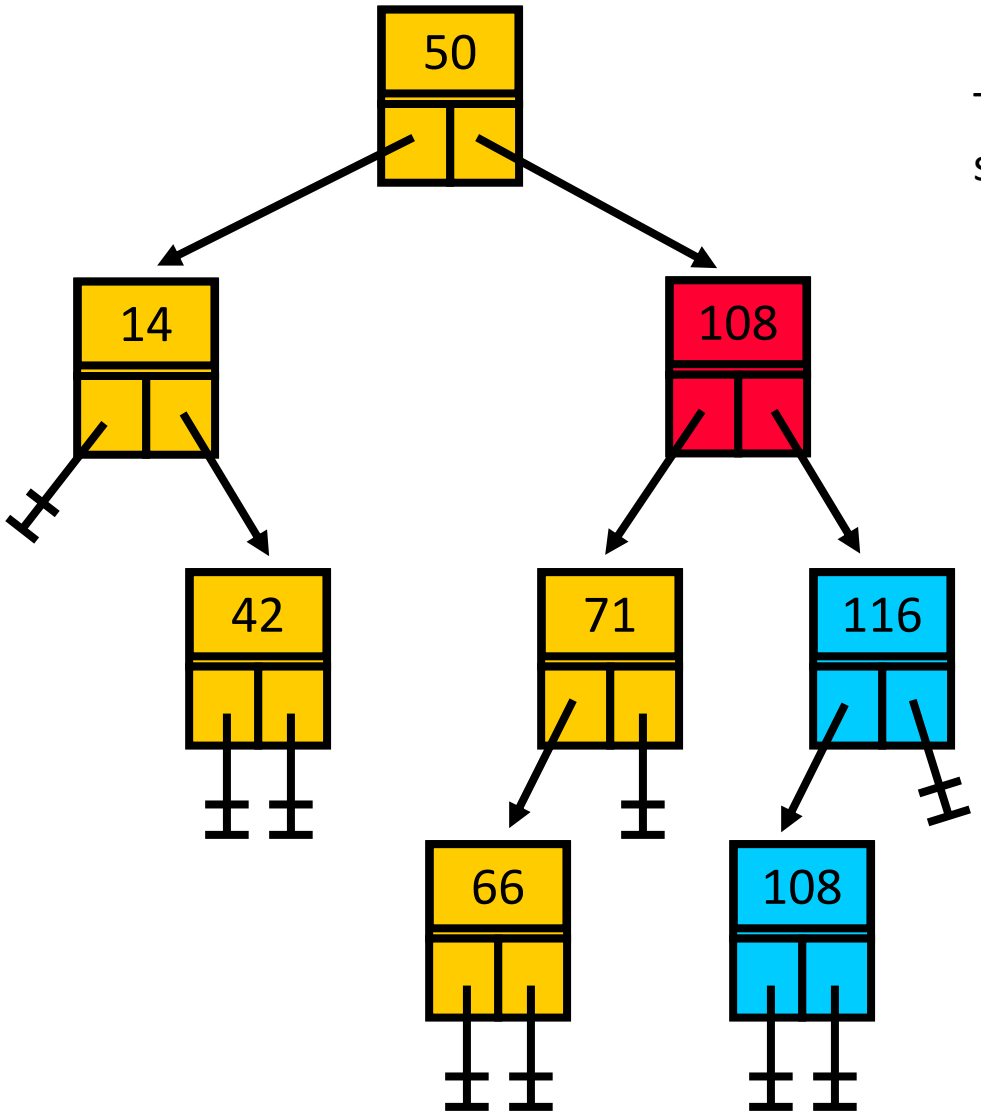
Look to the right sub-tree.

Delete a Node with Two Children



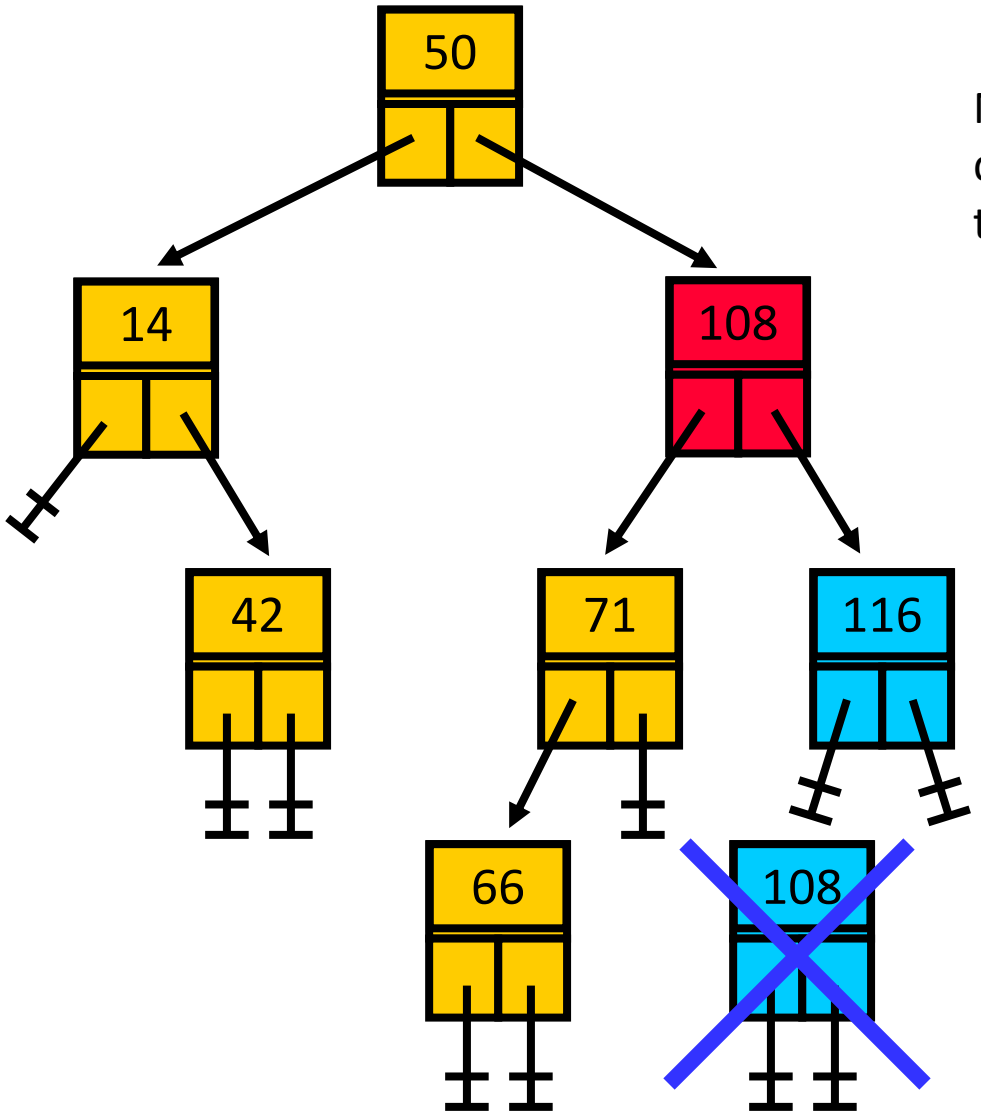
Find and copy the smallest value (this will erase the old value but creates a duplicate).

Delete a Node with Two Children



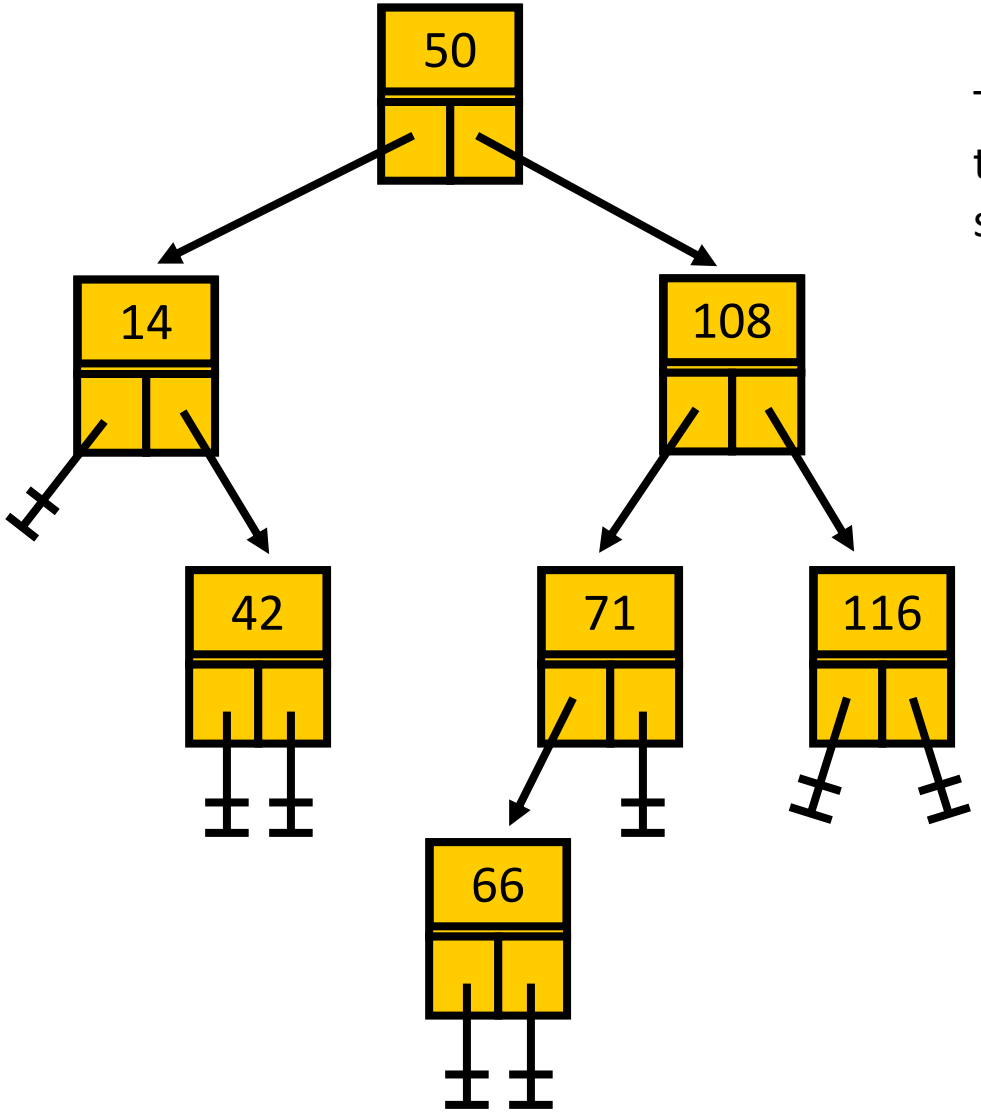
The resulting tree so far.

Delete a Node with Two Children



Now delete the duplicate from the left sub-tree.

Delete a Node with Two Children



The final resulting tree – still has search structure.

Summary

- **Deleting a node from a binary search tree involves two steps:**
  - **Search for the element**
  - **Then perform the deletion**
- **We must preserve the search structure and only delete the element which matches.**
- **Four cases:**
  - **Deleting a leaf node**
  - **Deleting a node with only the left child**
  - **Deleting a node with only the right child**
  - **Deleting a node with both children**

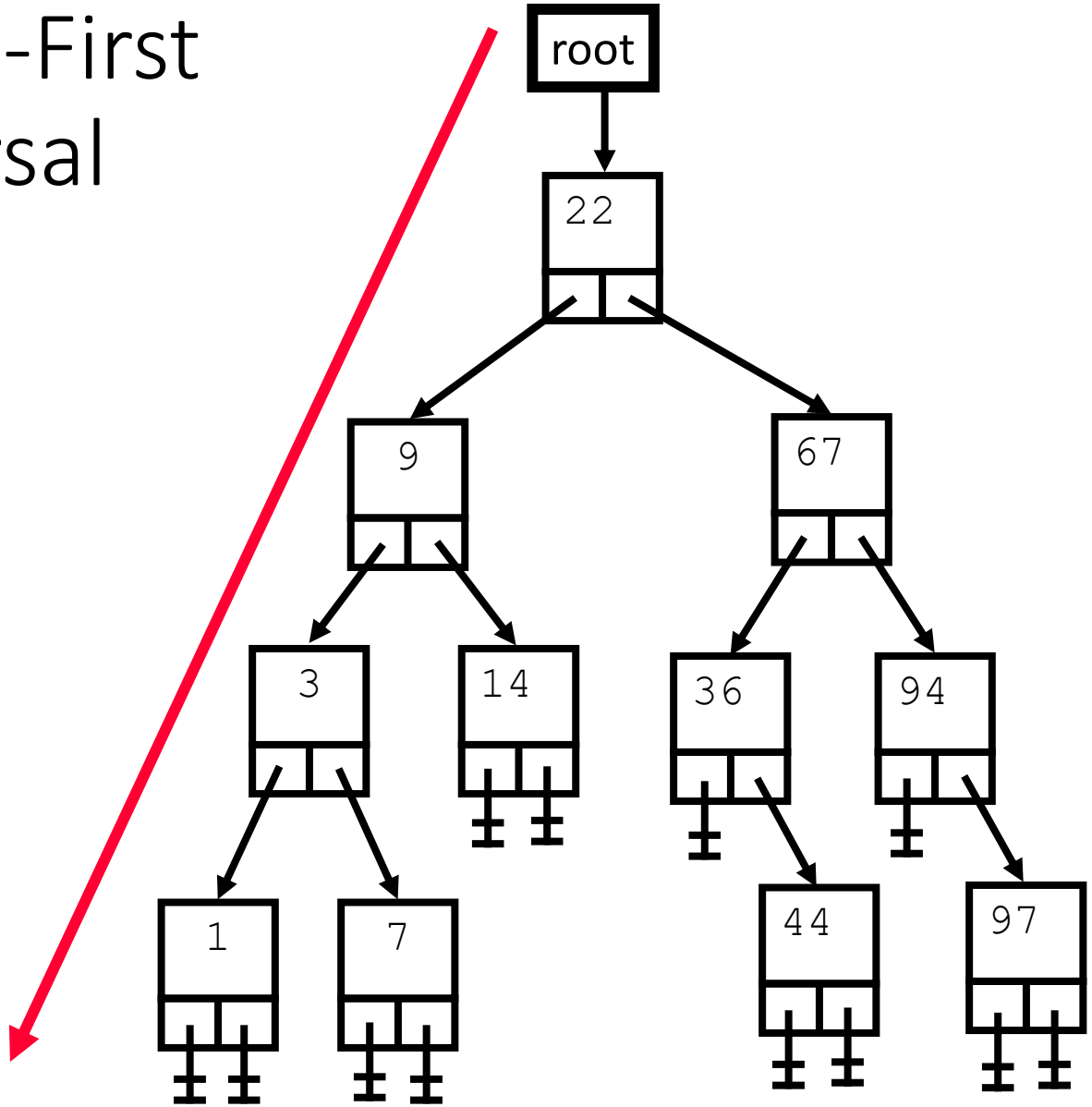
DEPTH FIRST SEARCH
and
BREADTH FIRST SEARCH

Depth vs. Breadth First Traversals

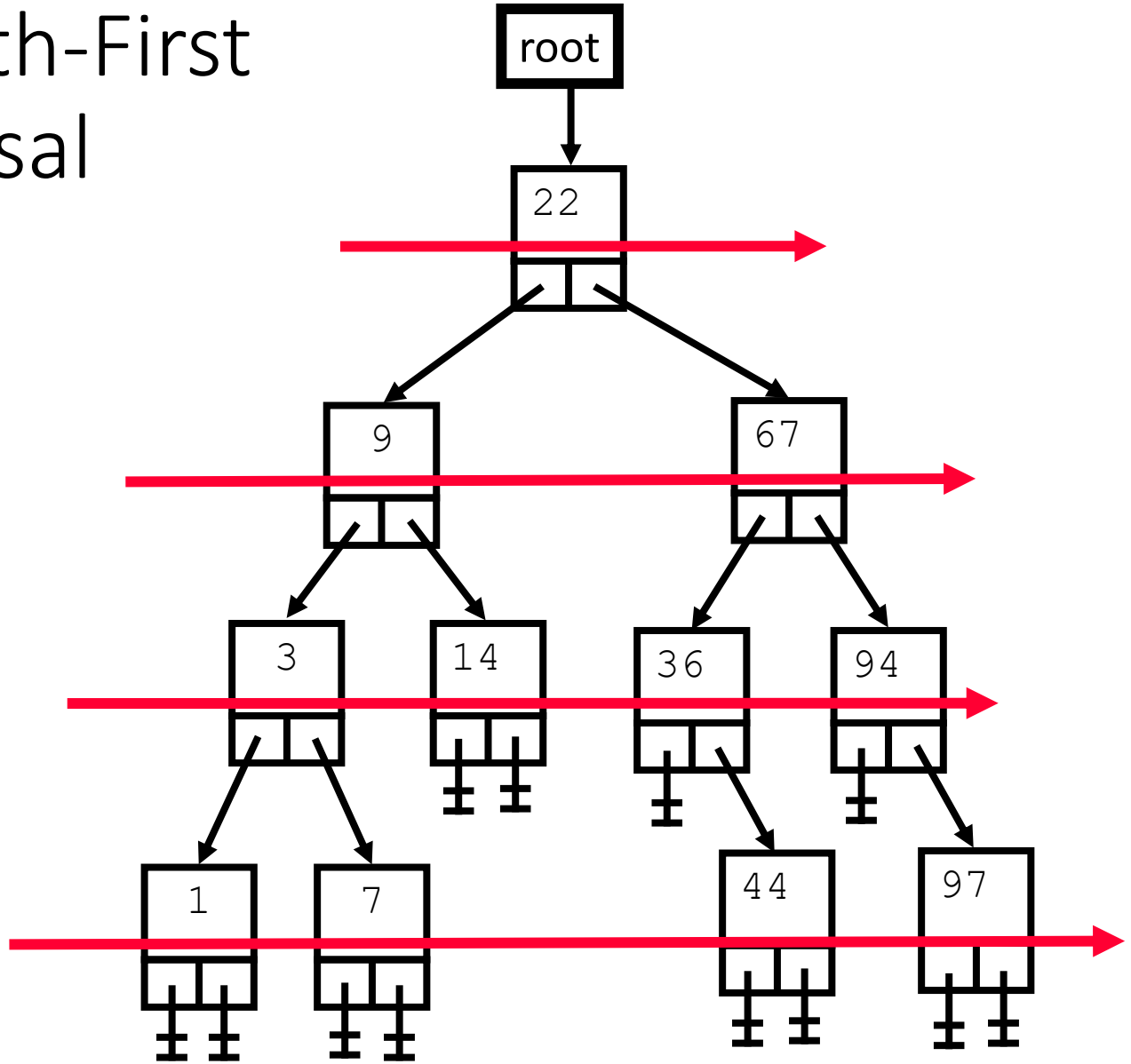
- **Depth First Traversals**
  - **Go down (deep)**
  - **In-, Pre-, Post-order**

- **Breadth First Traversals**
  - **Go across (shallow)**
  - **Require a queue to help**

Depth-First Traversal



Breadth-First Traversal



Depth First Traversal (In-Order)

```
Procedure DFT
    (current isoftype in Ptr toa Node)
// In order DFT
  if(current <> NIL) then
    DFT(current^.left)
    print(current^.data)
    DFT(Current^.right)
  endif
endprocedure
```

Depth First Traversal (Pre-Order)

```
Procedure DFT
    (current isoftype in Ptr toa Node)
// In order DFT
  if(current <> NIL) then
    DFT(current^.left)
    print(current^.data)
    DFT(Current^.right)
  endif
endprocedure
```

Move this here to make a Preorder DFT

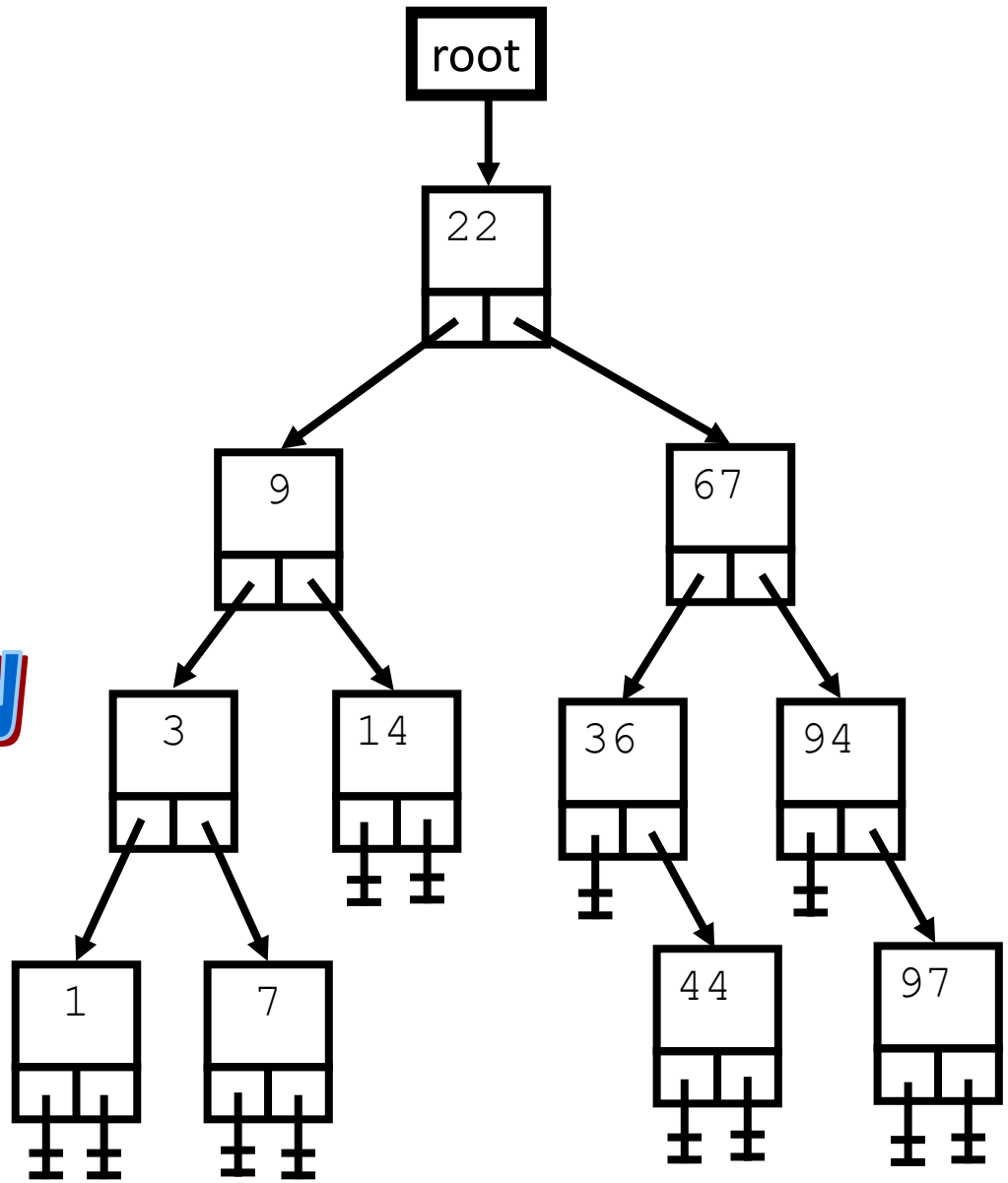
Depth First Traversal (Post-Order)

```
Procedure DFT
    (current isoftype in Ptr toa Node)
// In order DFT
  if(current <> NIL) then
    DFT(current^.left)
    print(current^.data)
    DFT(Current^.right)
  endif
endprocedure
```

Move this here to make a Postorder DFT

```
Proc DFT(pointer)
 pointer NOT NIL?
  DFT(left child)
  print(data)
  DFT(right child)
```

We have already seen these!



Breadth-First Traversal

Requires a queue to maintain which nodes to visit next.

Enqueue the root pointer

Loop until no elements in the queue

- **Dequeue(current)**
- **Enqueue current's left and right children**
- **Do work at current**

Breadth-First Traversal

```
Procedure BFT(root isoftype in Ptr toa Node)
 Q isoftype Queue
 Initialize(Q)
 temp isoftype Ptr toa Node
 OK isoftype Boolean
 enqueue(Q, root)
 // continued
```

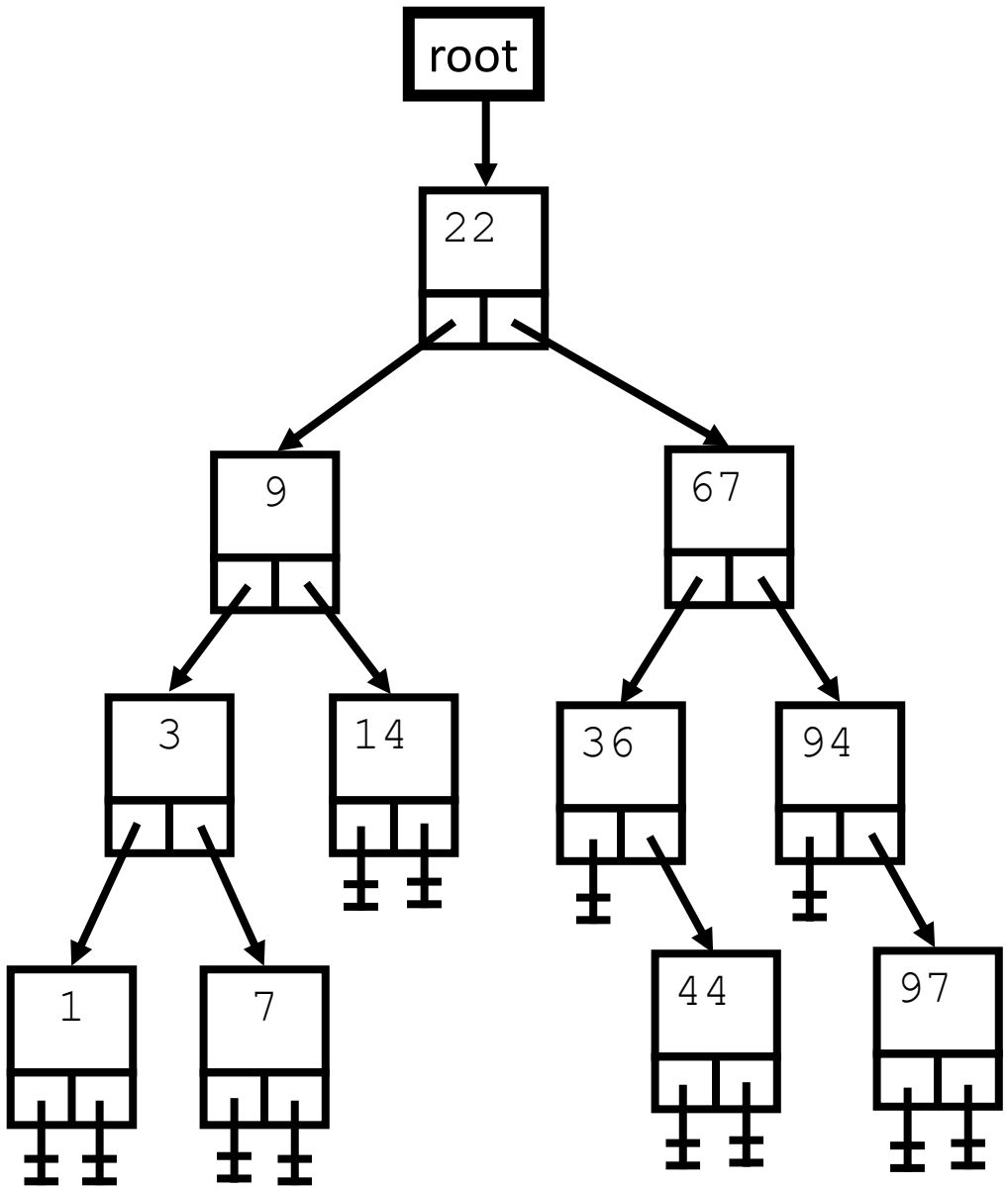
Breadth-First Traversal

```
 loop
   dequeue(temp, OK, Q)
   exitif(NOT OK)
   if(temp^.left <> NIL) then
    enqueue(temp^.left, Q))
   endif
   if(temp^.right <> NIL) then
    enqueue(temp^.right, Q)
   endif
   print(temp^.data) // processing node
 endloop
endprocedure
```

```
Enqueue(root)
loop
  exitif Q empty
  dequeue(aNode)
  enqueue(children)
  print(data)
endloop
```

Q:

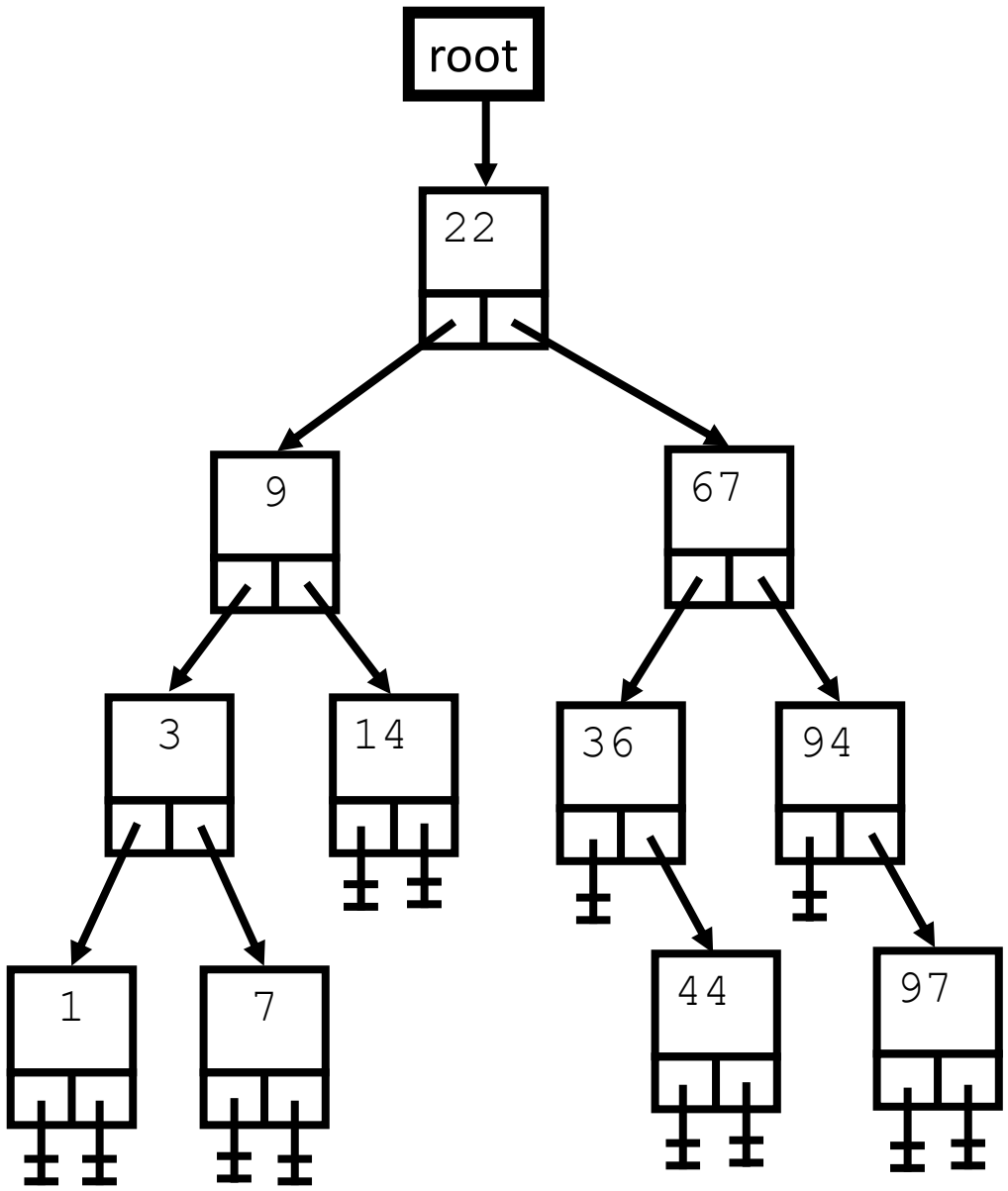
aNode:



```
Enqueue(root)
loop
  exitif Q empty
  dequeue(aNode)
  enqueue(children)
  print(data)
endloop
```

Q:22

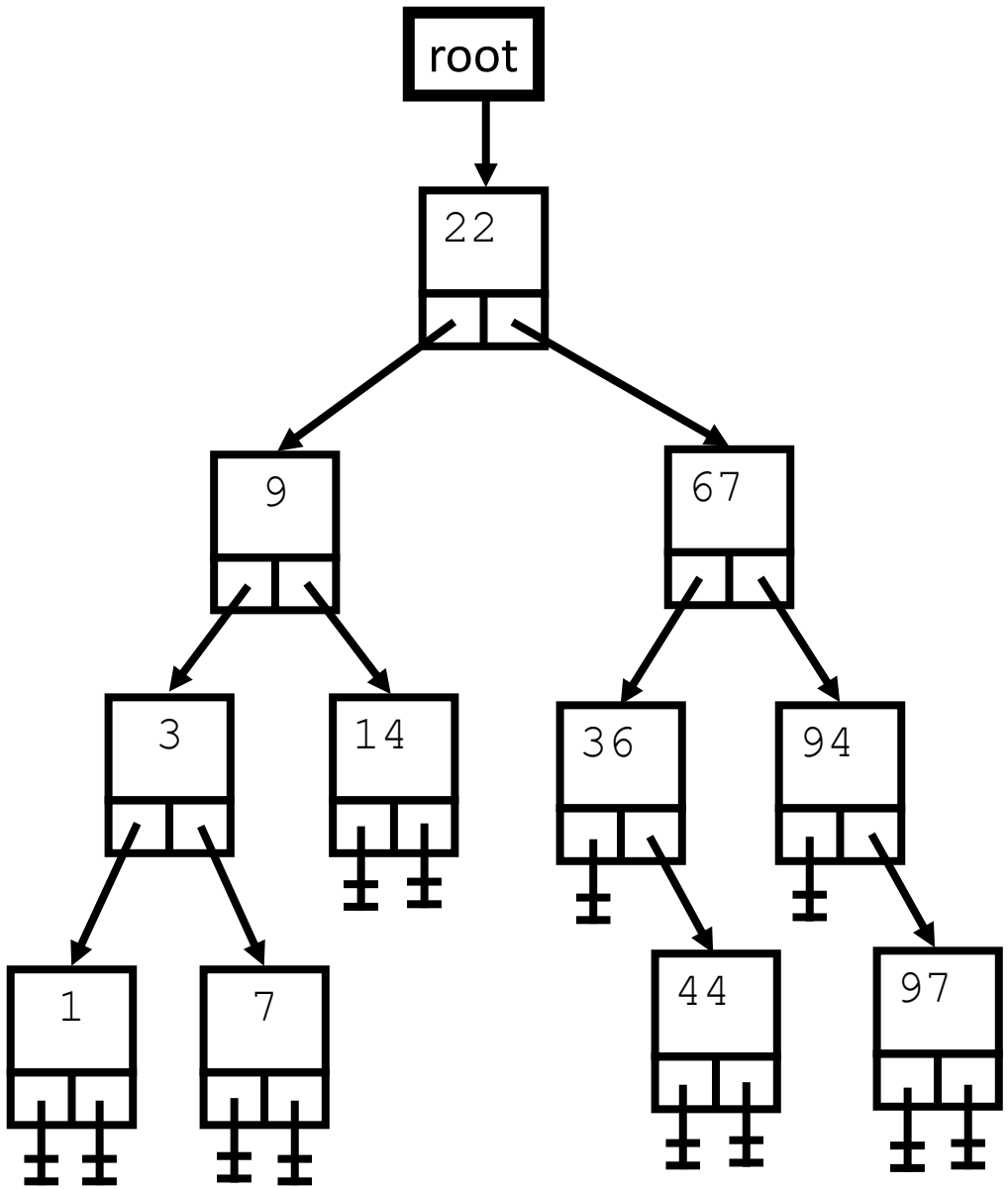
aNode:



```
Enqueue(root)
loop
  exitif Q empty
  dequeue(aNode)
  enqueue(children)
  print(data)
endloop
```

Q:22

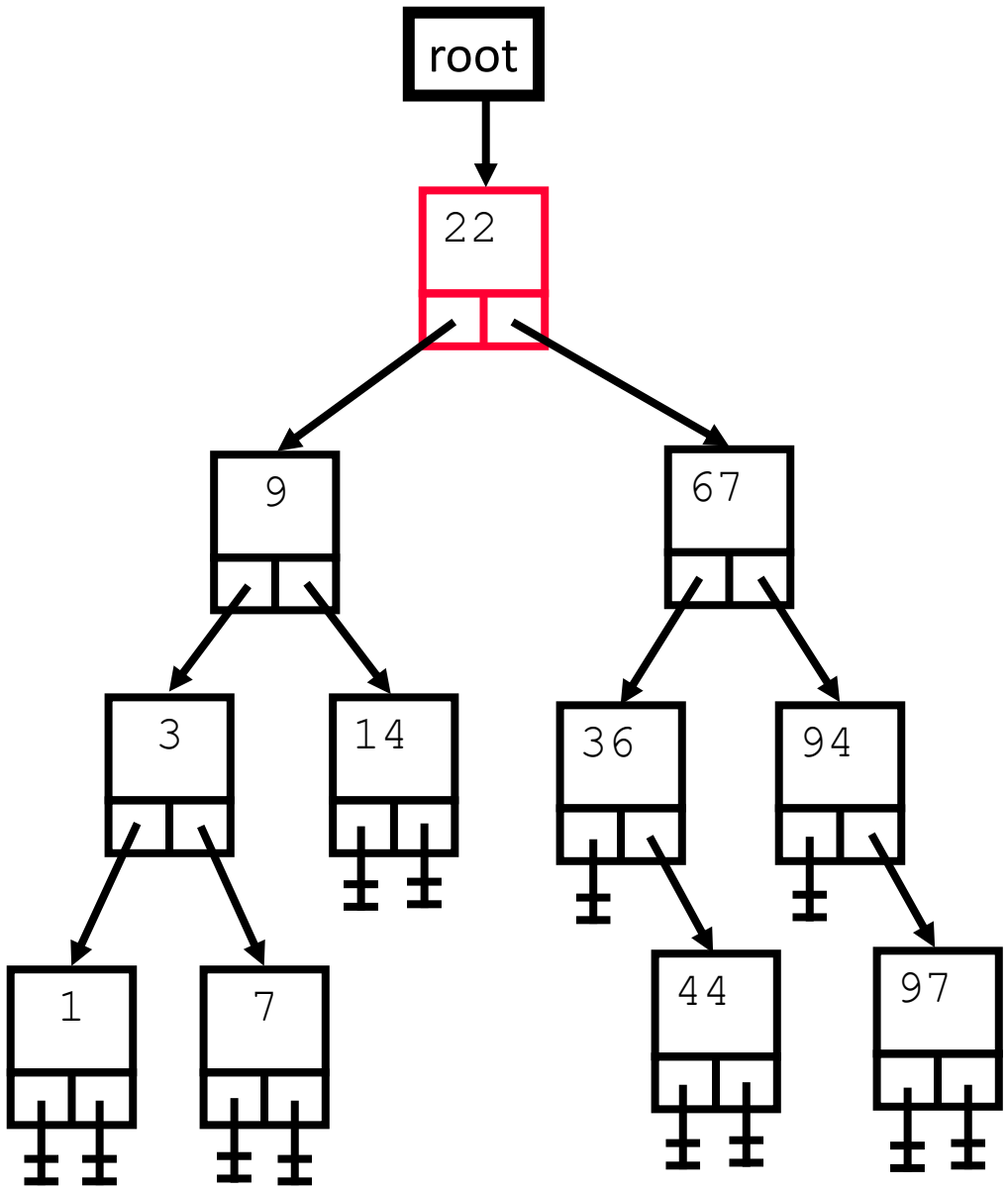
aNode:



```
Enqueue(root)
loop
  exitif Q empty
  dequeue(aNode)
  enqueue(children)
  print(aNode.data)
endloop
```

Q:

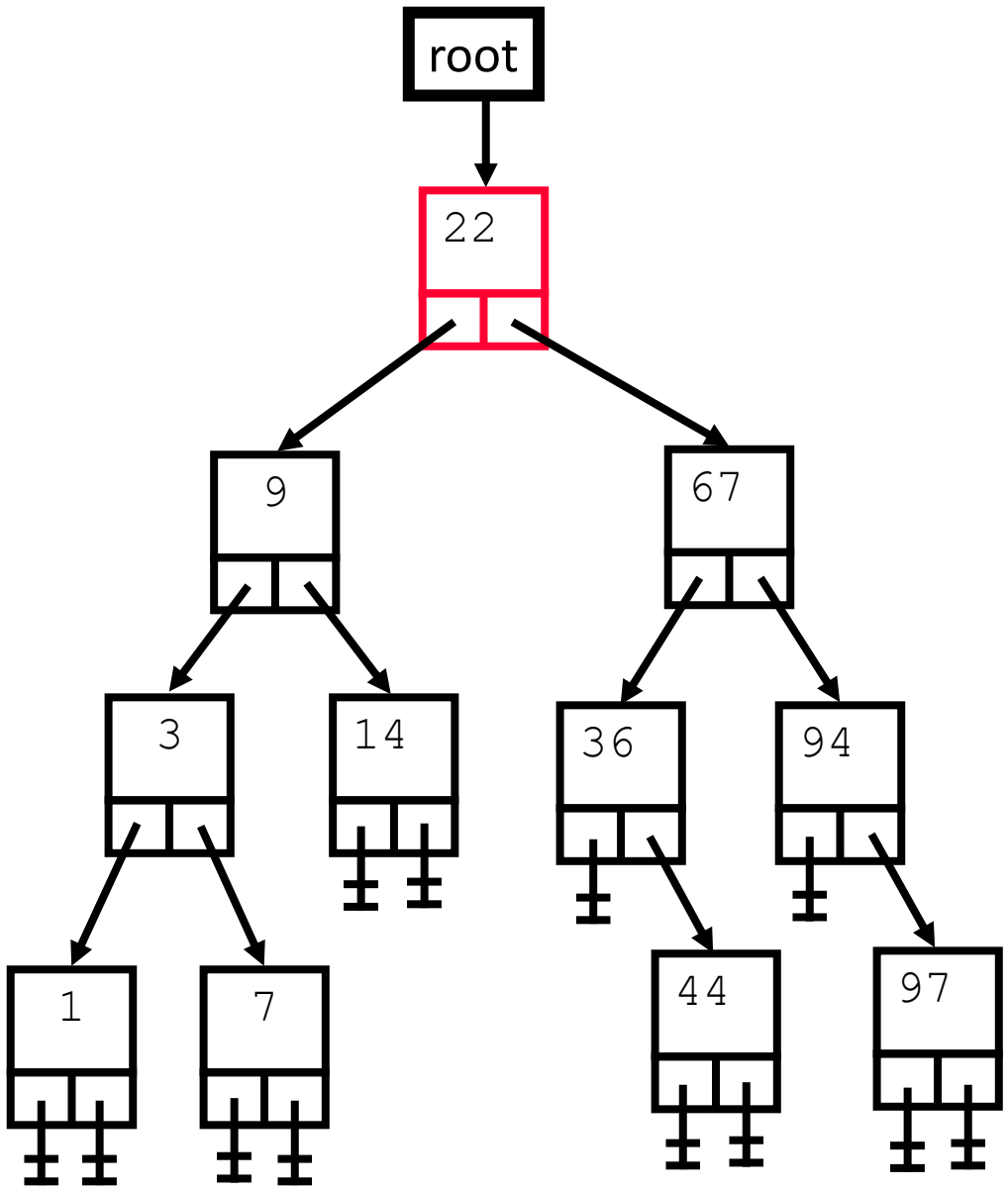
aNode:22



```
Enqueue(root)
loop
  exitif Q empty
  dequeue(aNode)
  enqueue(children)
  print(aNode.data)
endloop
```

Q:9 67

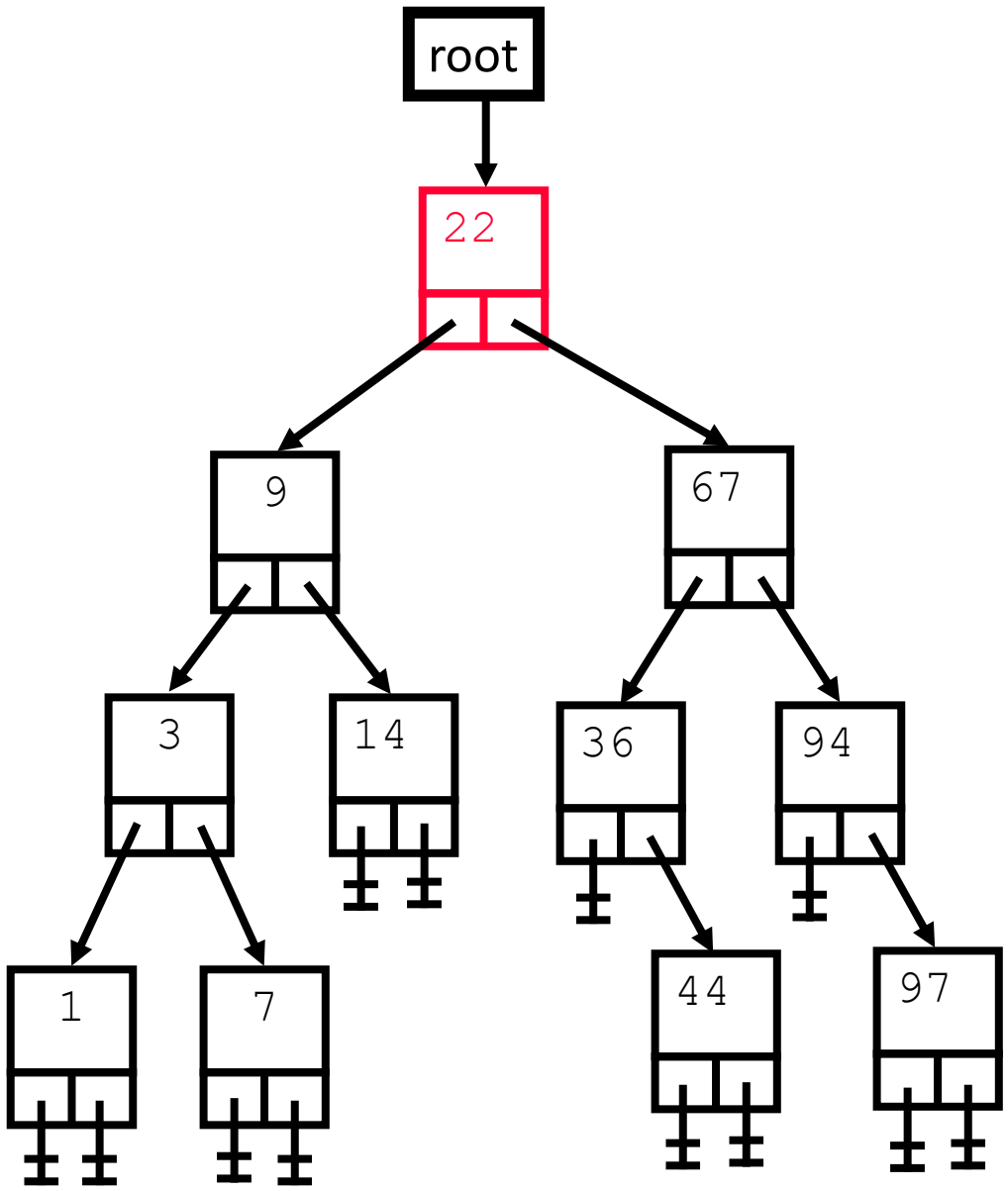
aNode:22



```
Enqueue(root)
loop
  exitif Q empty
  dequeue(aNode)
  enqueue(children)
  print(aNode.data)
endloop
```

Q:9 67

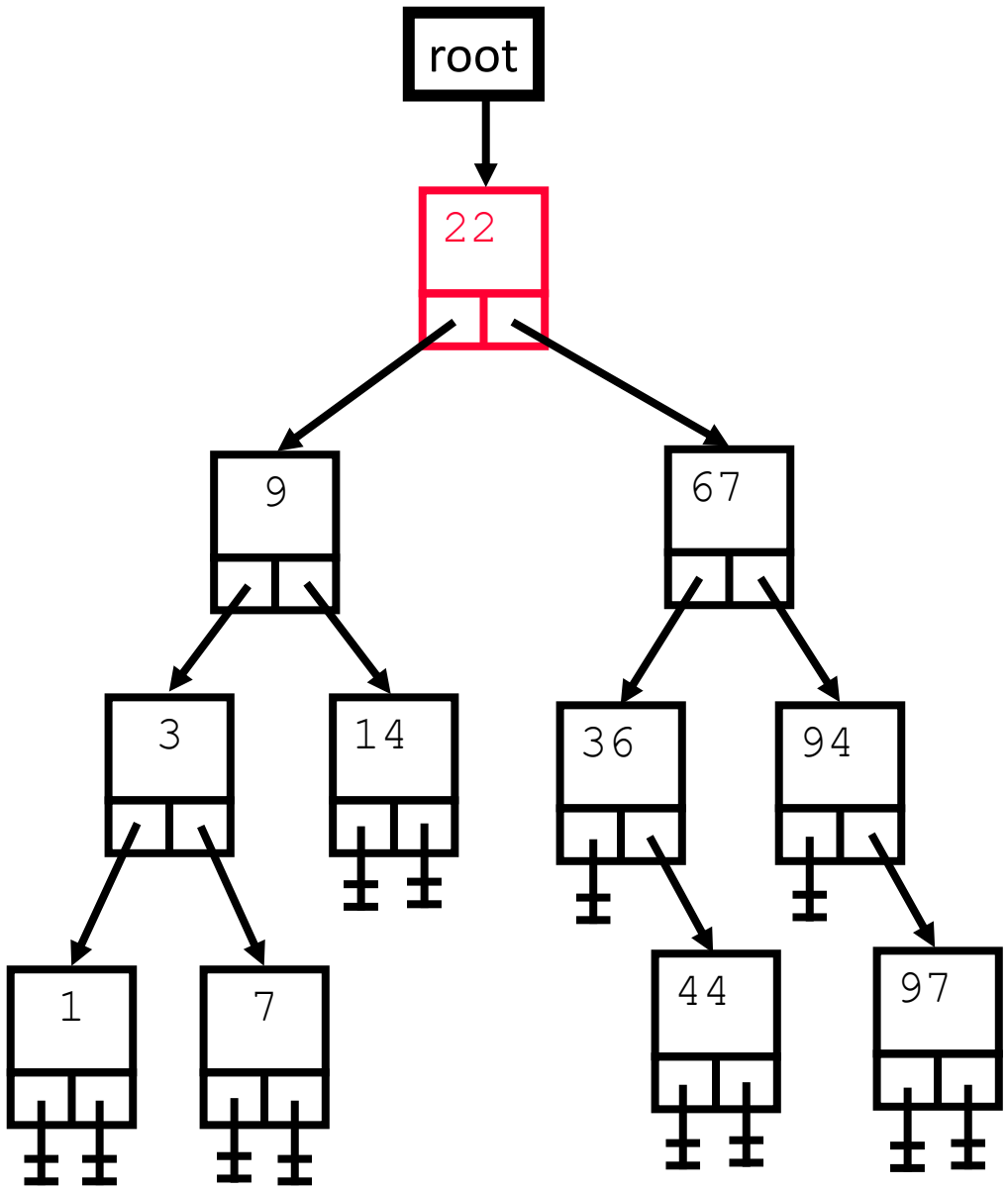
aNode:22



```
Enqueue(root)
loop
  exitif Q empty
  dequeue(aNode)
  enqueue(children)
  print(aNode.data)
endloop
```

Q:9 67

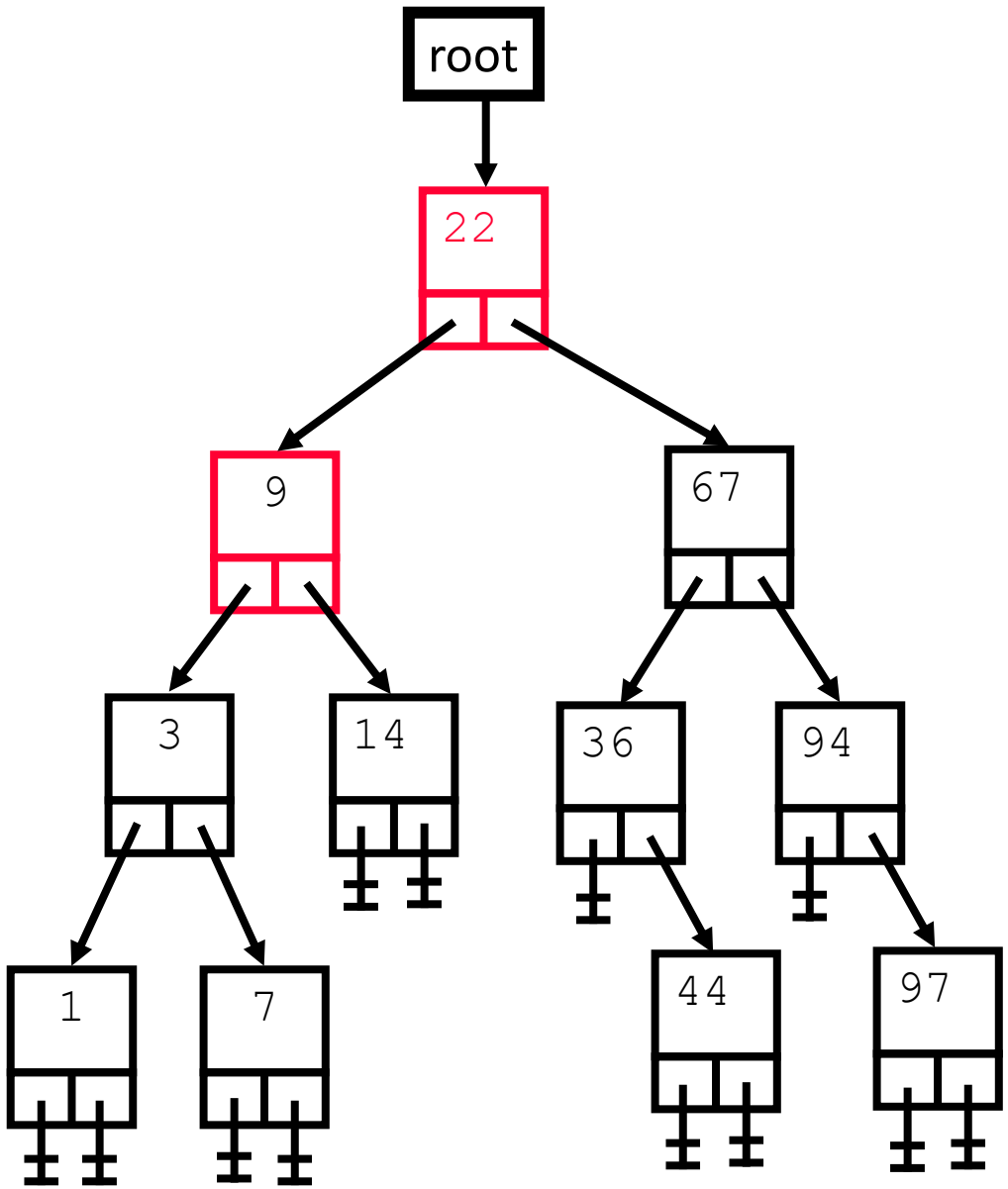
aNode:22



```
Enqueue(root)
loop
  exitif Q empty
  dequeue(aNode)
  enqueue(children)
  print(aNode.data)
endloop
```

Q:67

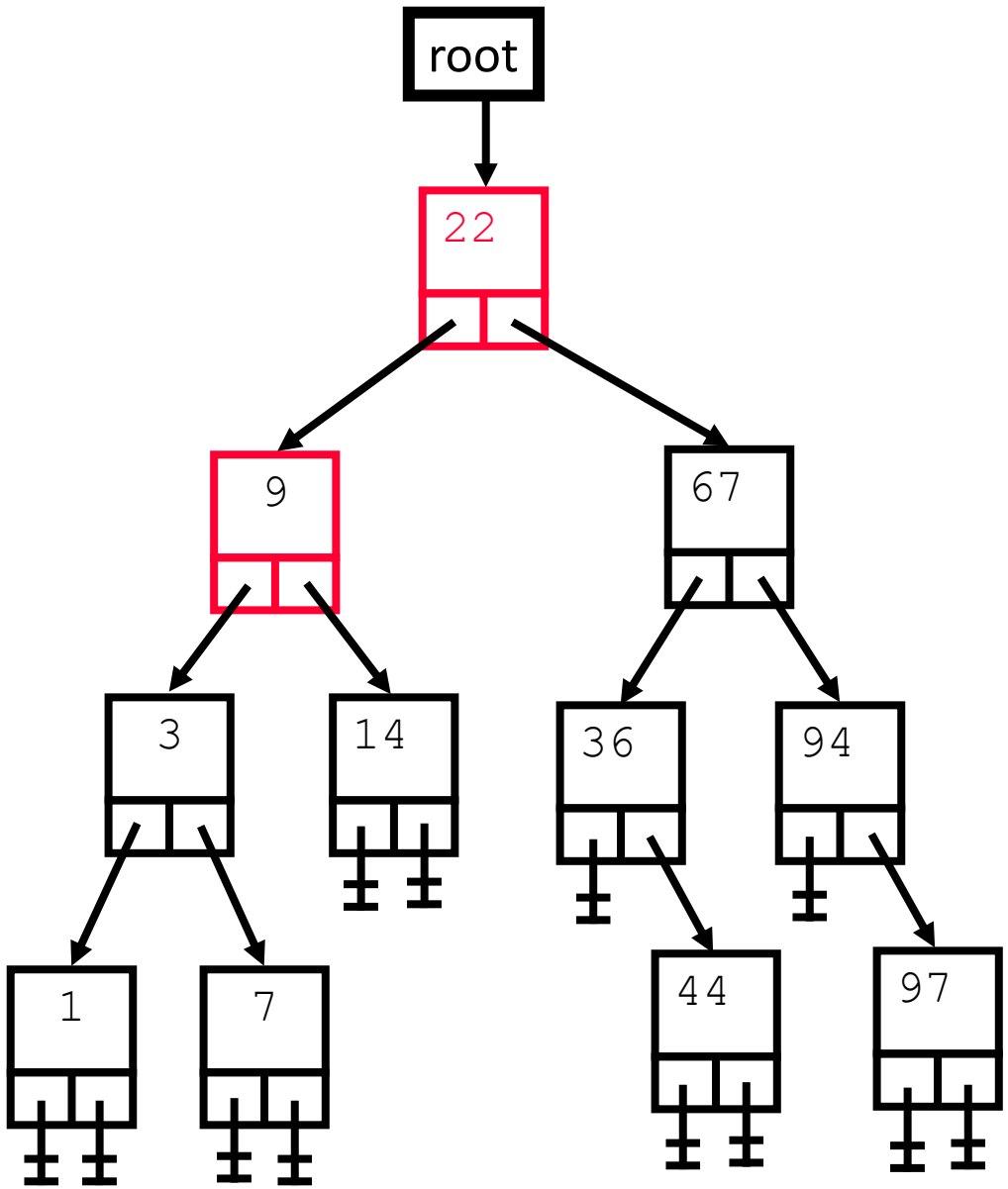
aNode:9



```
Enqueue(root)
loop
  exitif Q empty
  dequeue(aNode)
  enqueue(children)
  print(aNode.data)
endloop
```

Q:67 3 14

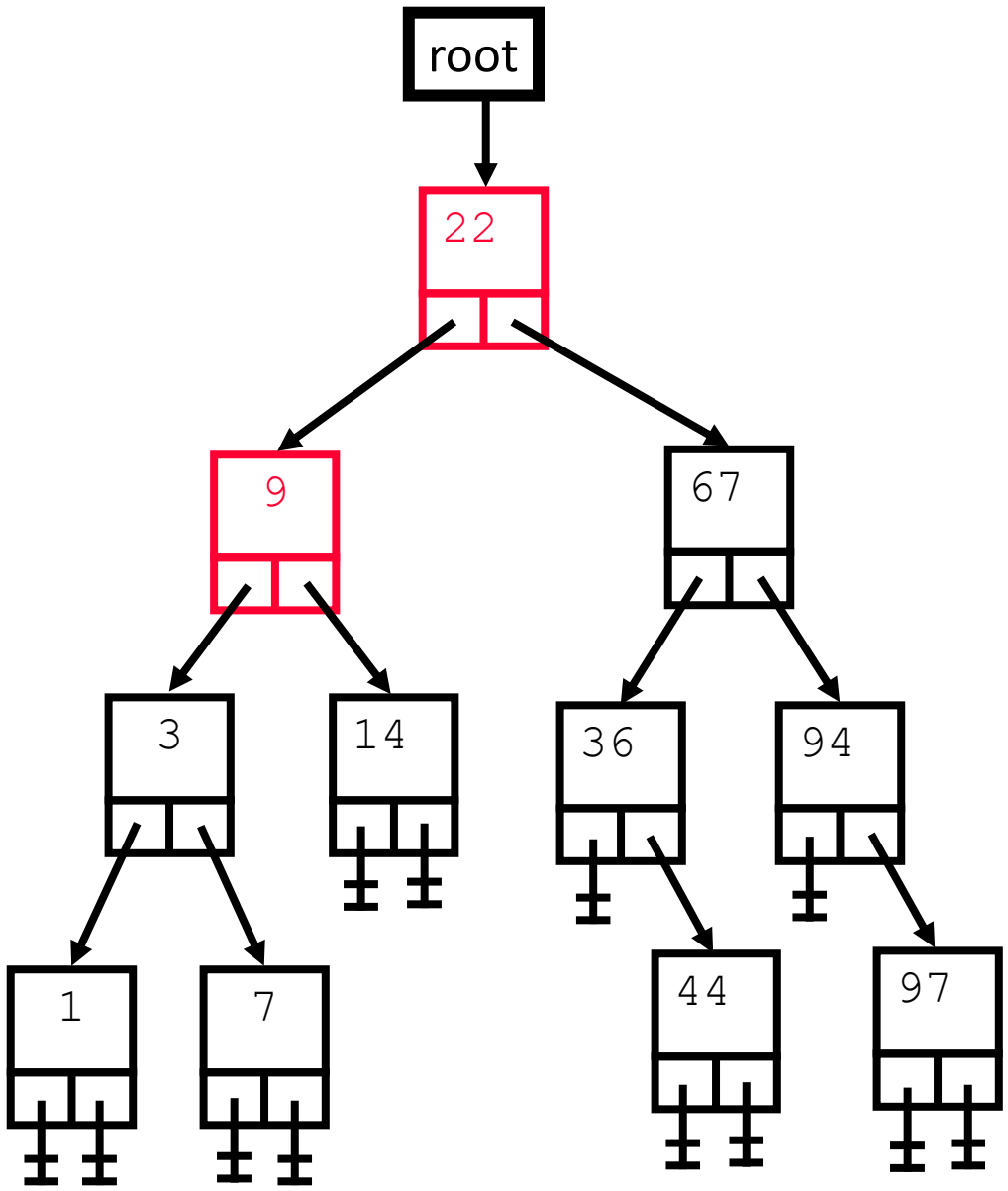
aNode:9



```
Enqueue(root)
loop
  exitif Q empty
  dequeue(aNode)
  enqueue(children)
  print(aNode.data)
endloop
```

Q:67 3 14

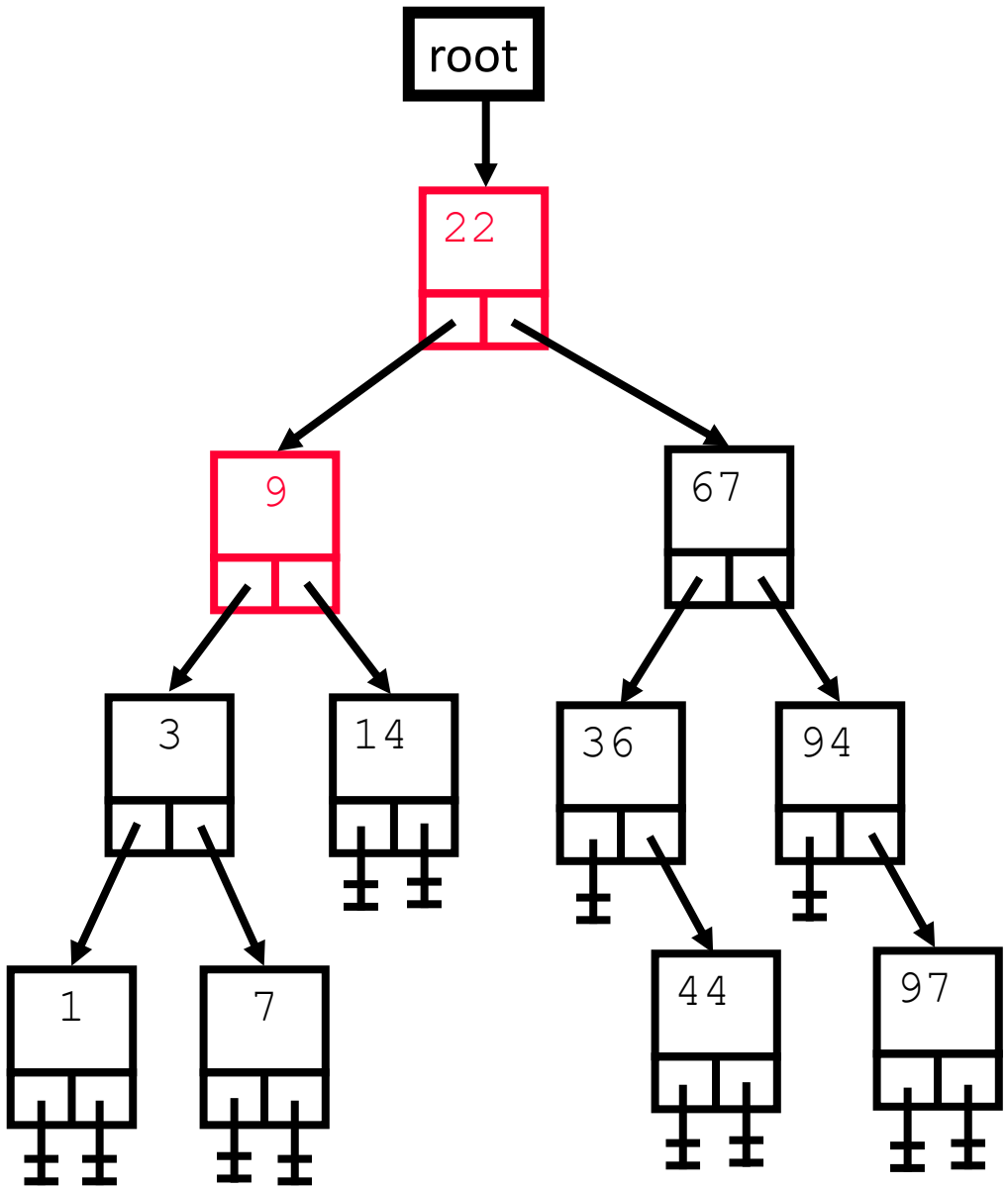
aNode:9



```
Enqueue(root)
loop
  exitif Q empty
  dequeue(aNode)
  enqueue(children)
  print(aNode.data)
endloop
```

Q:67 3 14

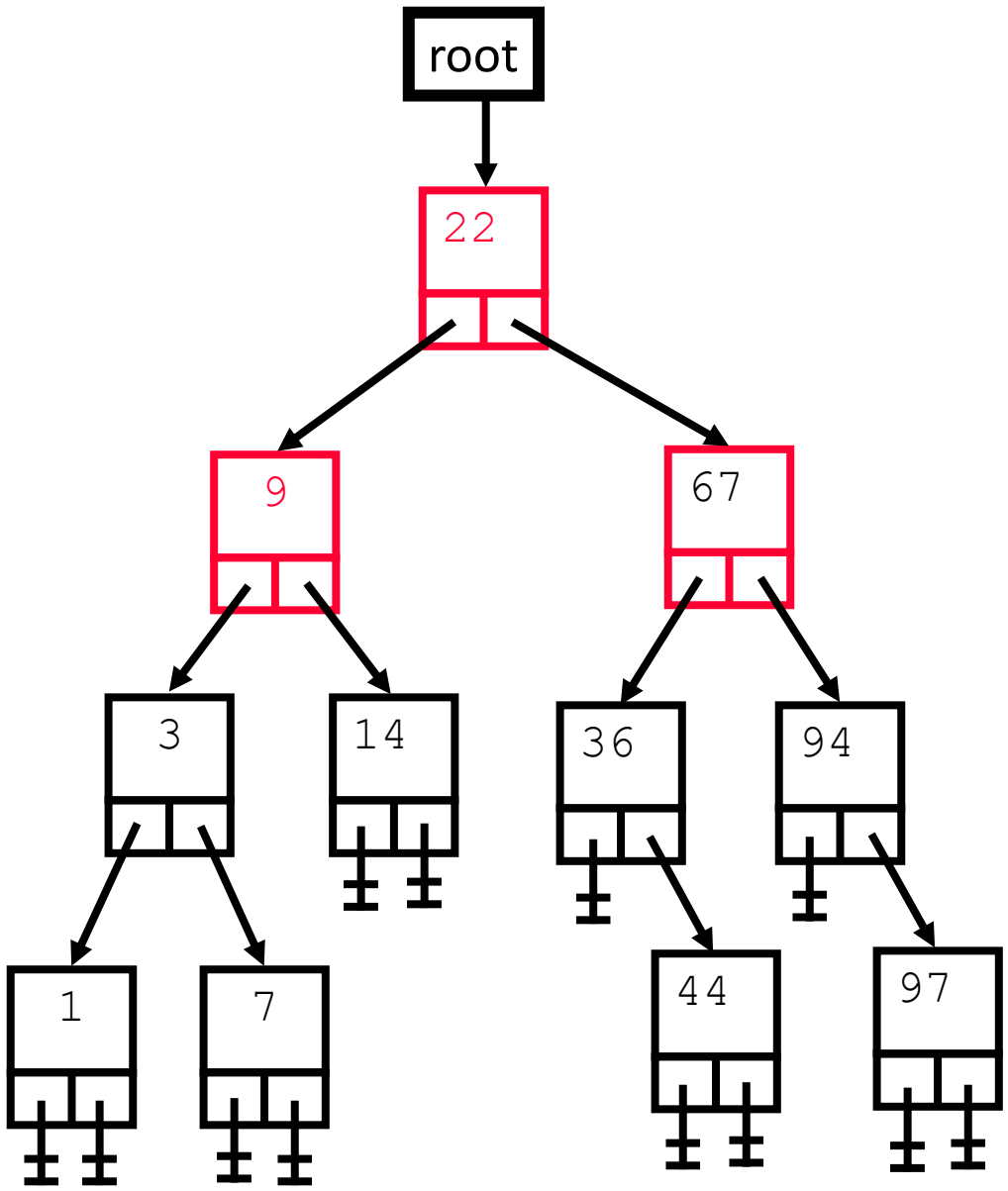
aNode:9



```
Enqueue(root)
loop
  exitif Q empty
  dequeue(aNode)
  enqueue(children)
  print(aNode.data)
endloop
```

Q:3 14

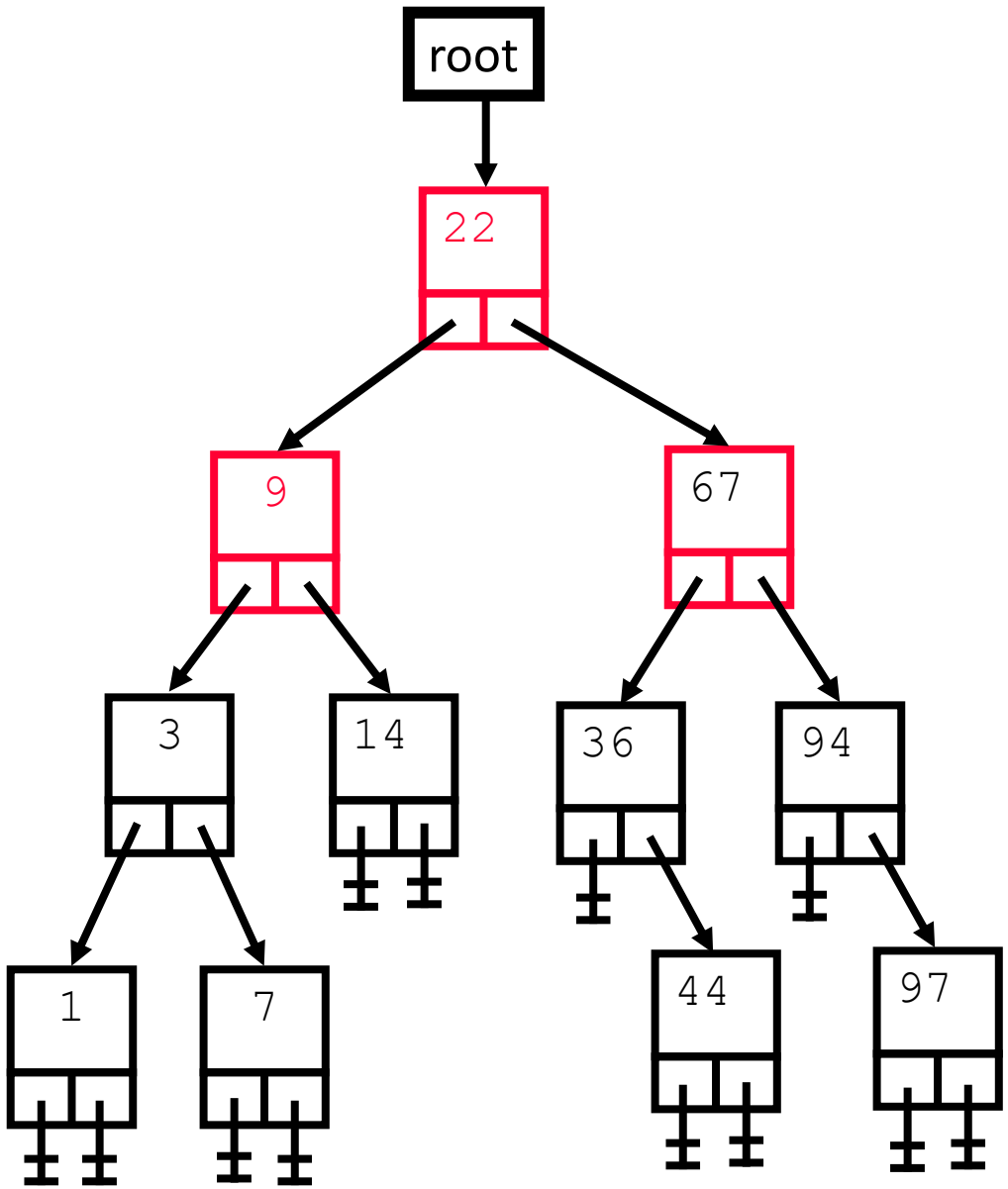
aNode:67



```
Enqueue(root)
loop
  exitif Q empty
  dequeue(aNode)
  enqueue(children)
  print(aNode.data)
endloop
```

Q:3 14 36 94

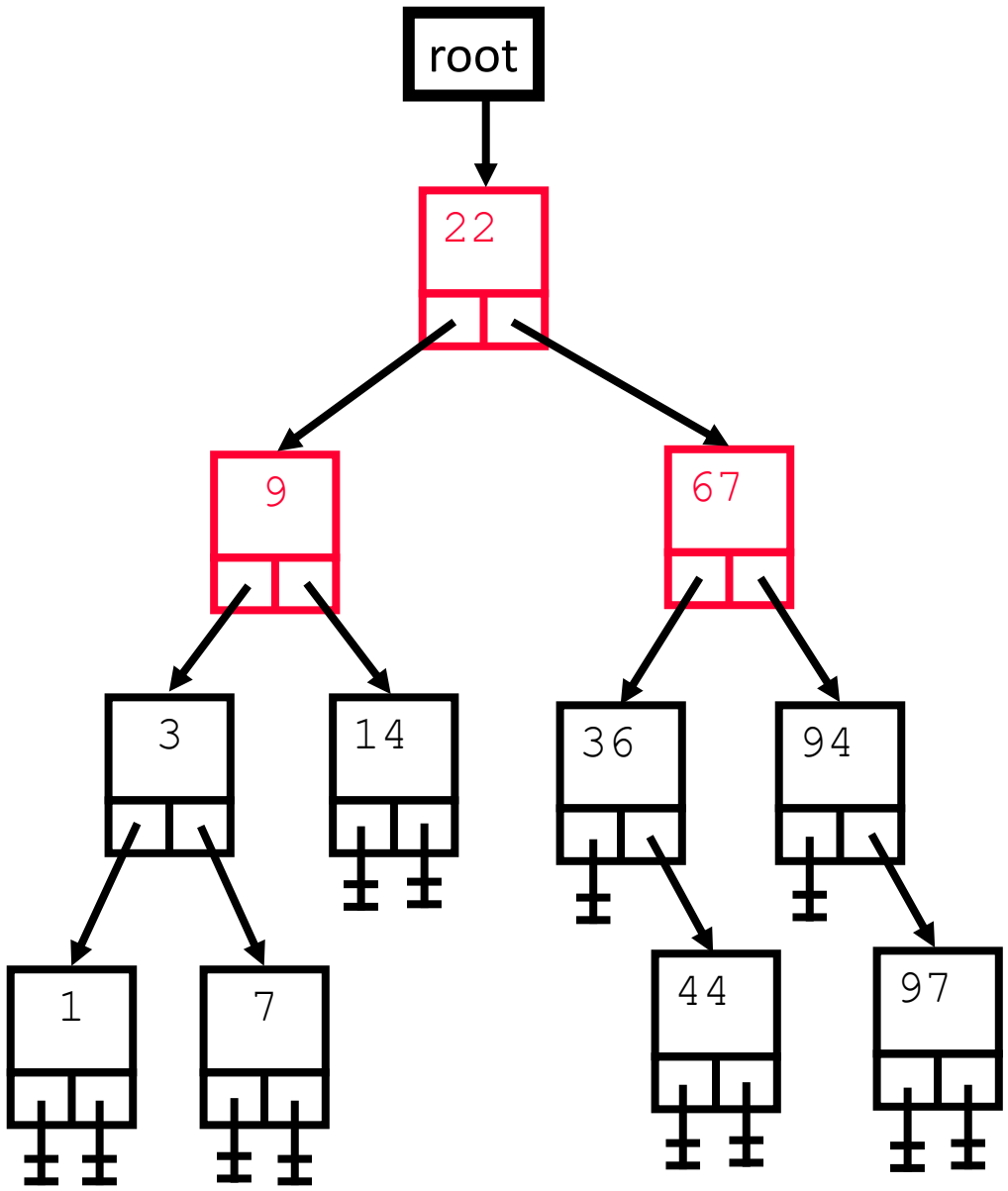
aNode:67



```
Enqueue(root)
loop
  exitif Q empty
  dequeue(aNode)
  enqueue(children)
  print(aNode.data)
endloop
```

Q:3 14 36 94

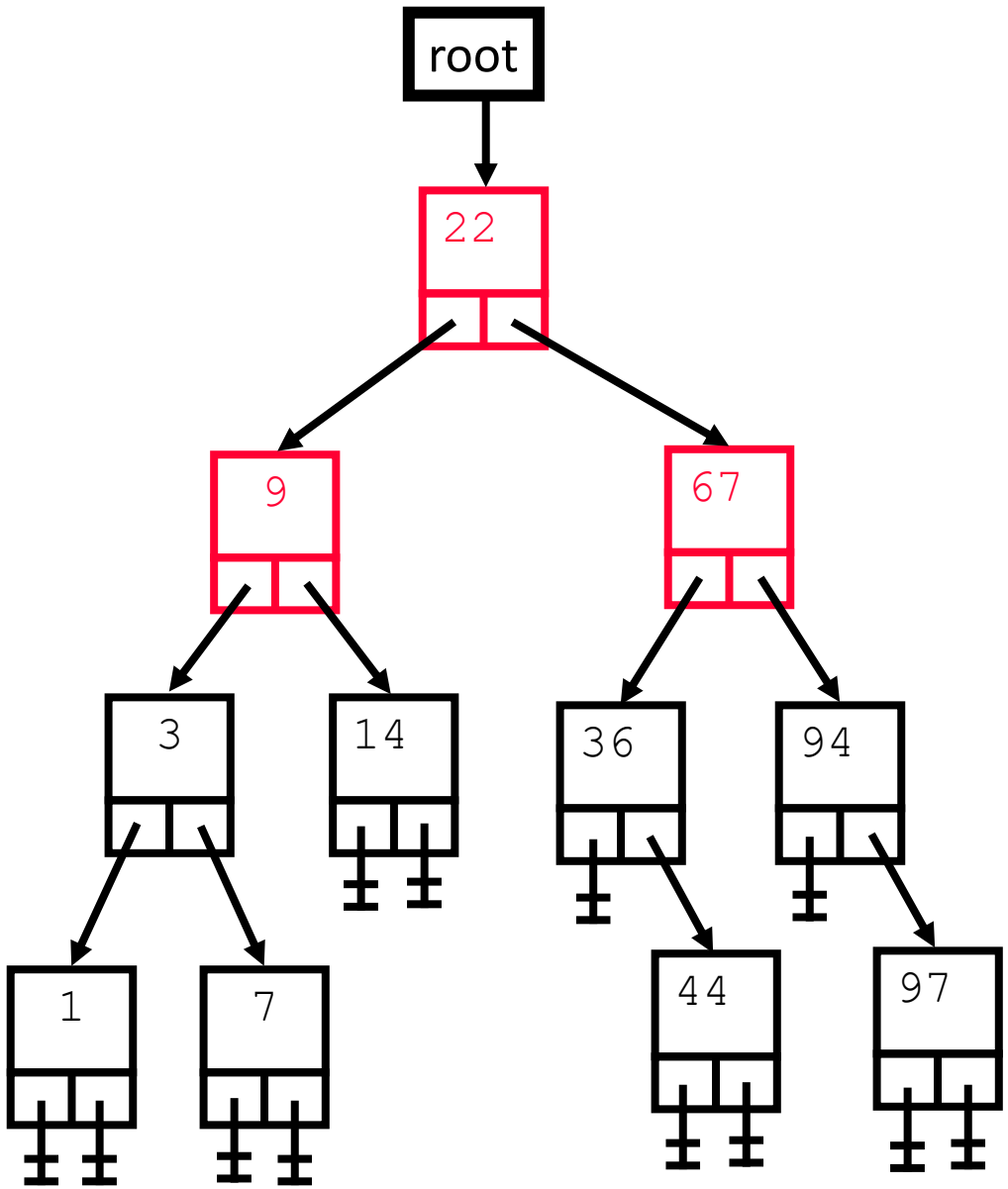
aNode:67



```
Enqueue(root)
loop
  exitif Q empty
  dequeue(aNode)
  enqueue(children)
  print(aNode.data)
endloop
```

Q:3 14 36 94

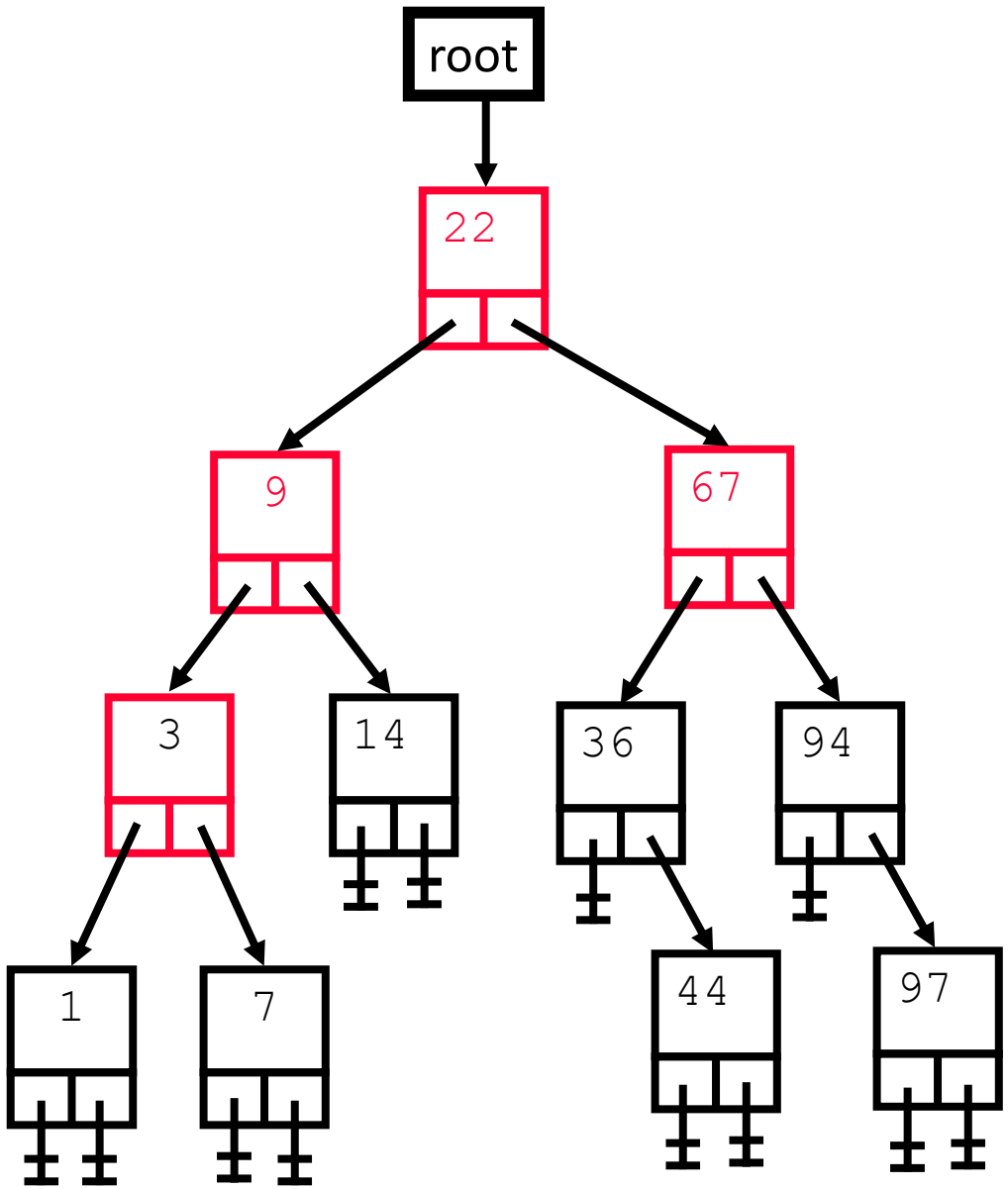
aNode:67



```
Enqueue(root)
loop
  exitif Q empty
  dequeue(aNode)
  enqueue(children)
  print(aNode.data)
endloop
```

Q:14 36 94

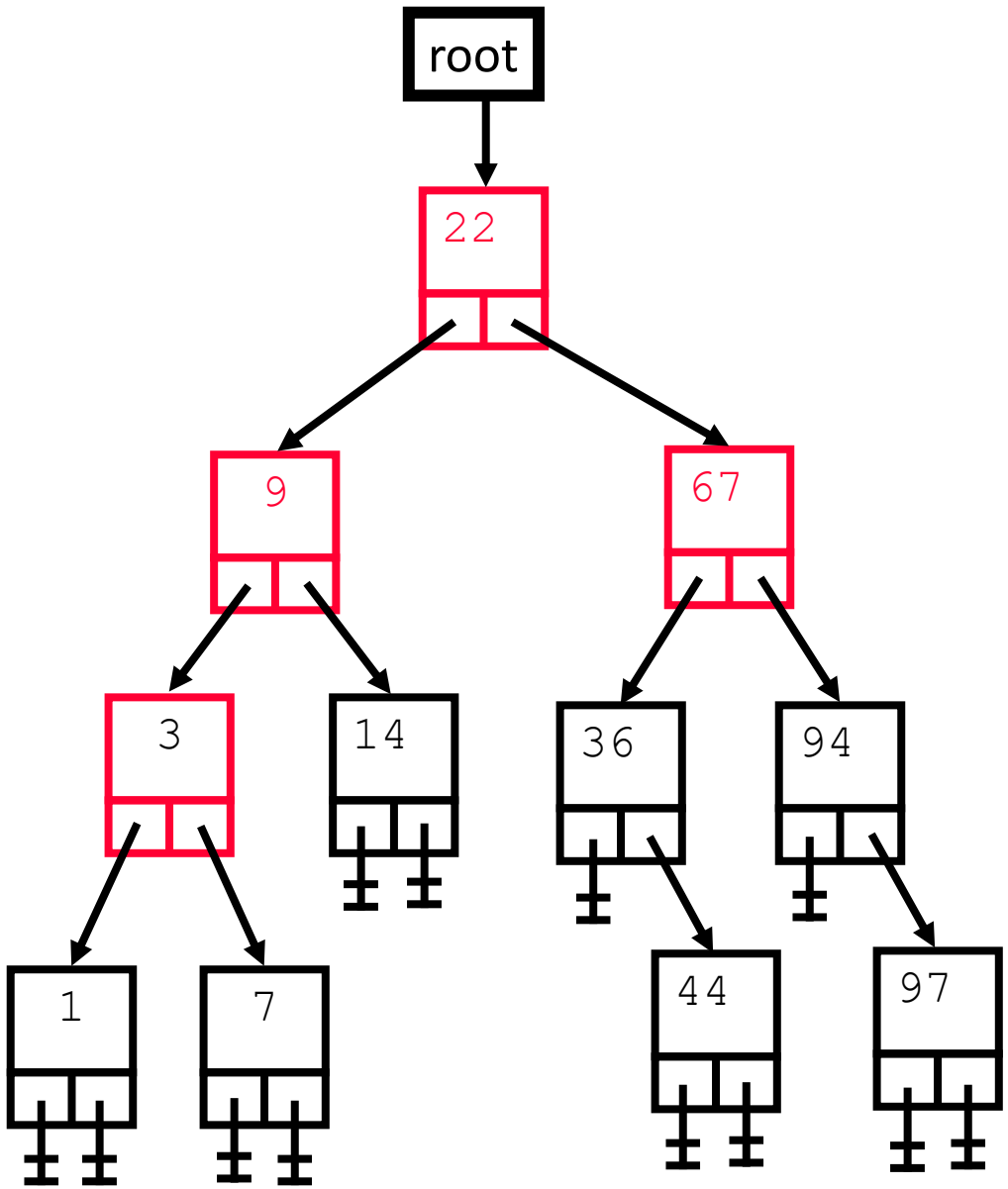
aNode:3



```
Enqueue(root)
loop
  exitif Q empty
  dequeue(aNode)
  enqueue(children)
  print(aNode.data)
endloop
```

Q:14 36 94 1 7

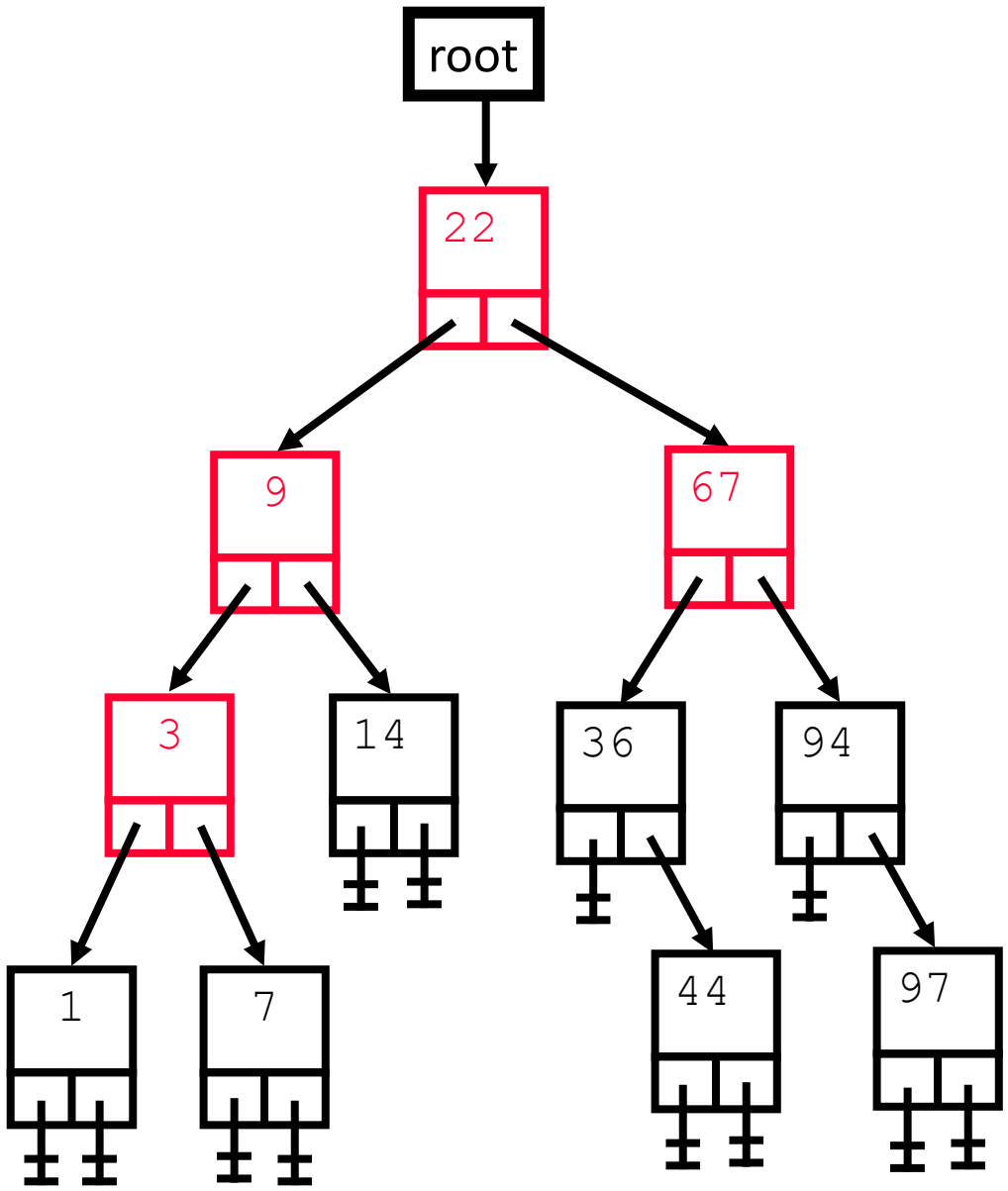
aNode:3



```
Enqueue(root)
loop
  exitif Q empty
  dequeue(aNode)
  enqueue(children)
  print(aNode.data)
endloop
```

Q:14 36 94 1 7

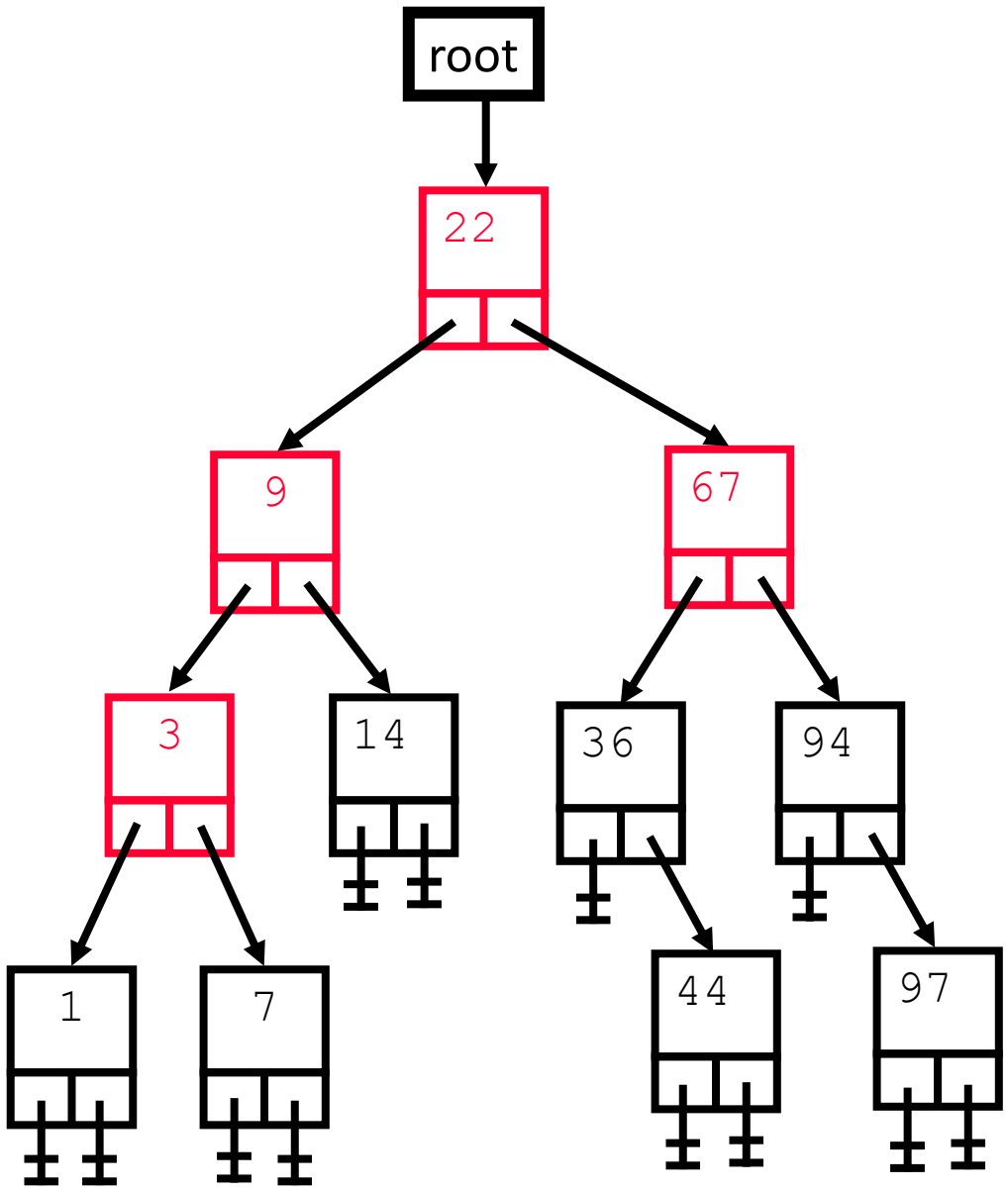
aNode:3



```
Enqueue(root)
loop
  exitif Q empty
  dequeue(aNode)
  enqueue(children)
  print(aNode.data)
endloop
```

Q:14 36 94 1 7

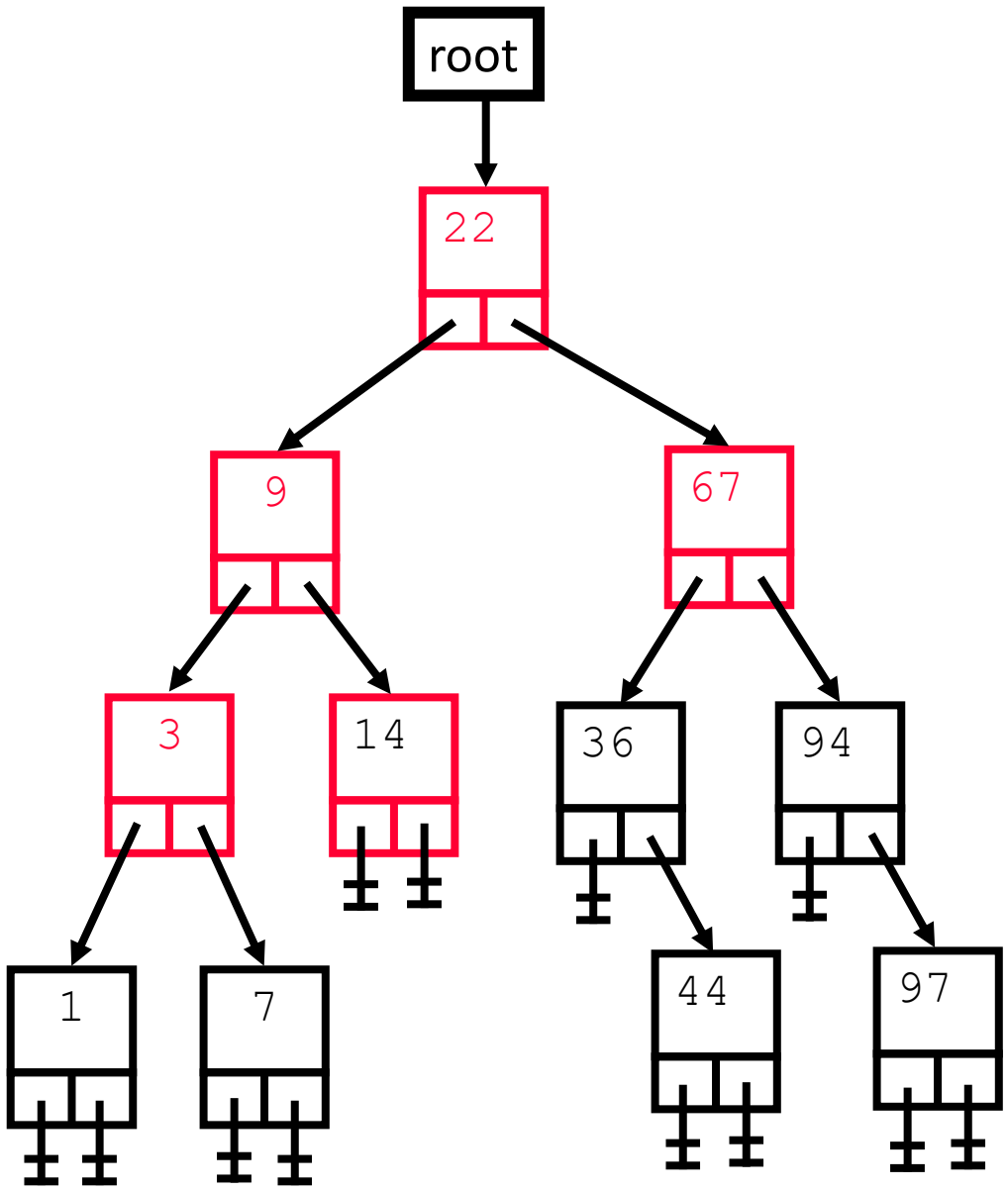
aNode:3



```
Enqueue(root)
loop
  exitif Q empty
  dequeue(aNode)
  enqueue(children)
  print(aNode.data)
endloop
```

Q:36 94 1 7

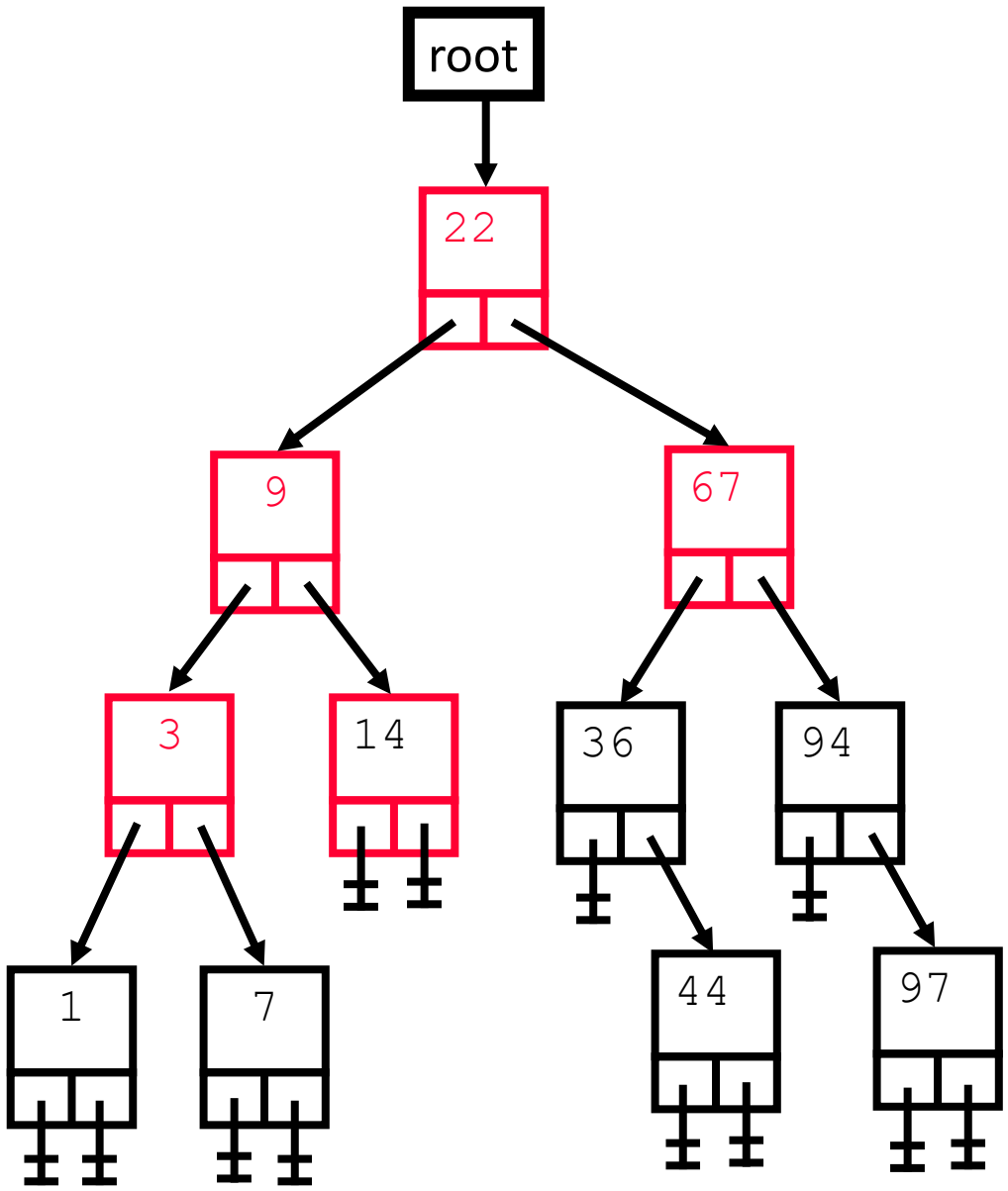
aNode:14



```
Enqueue(root)
loop
  exitif Q empty
  dequeue(aNode)
  enqueue(children)
  print(aNode.data)
endloop
```

Q:36 94 1 7

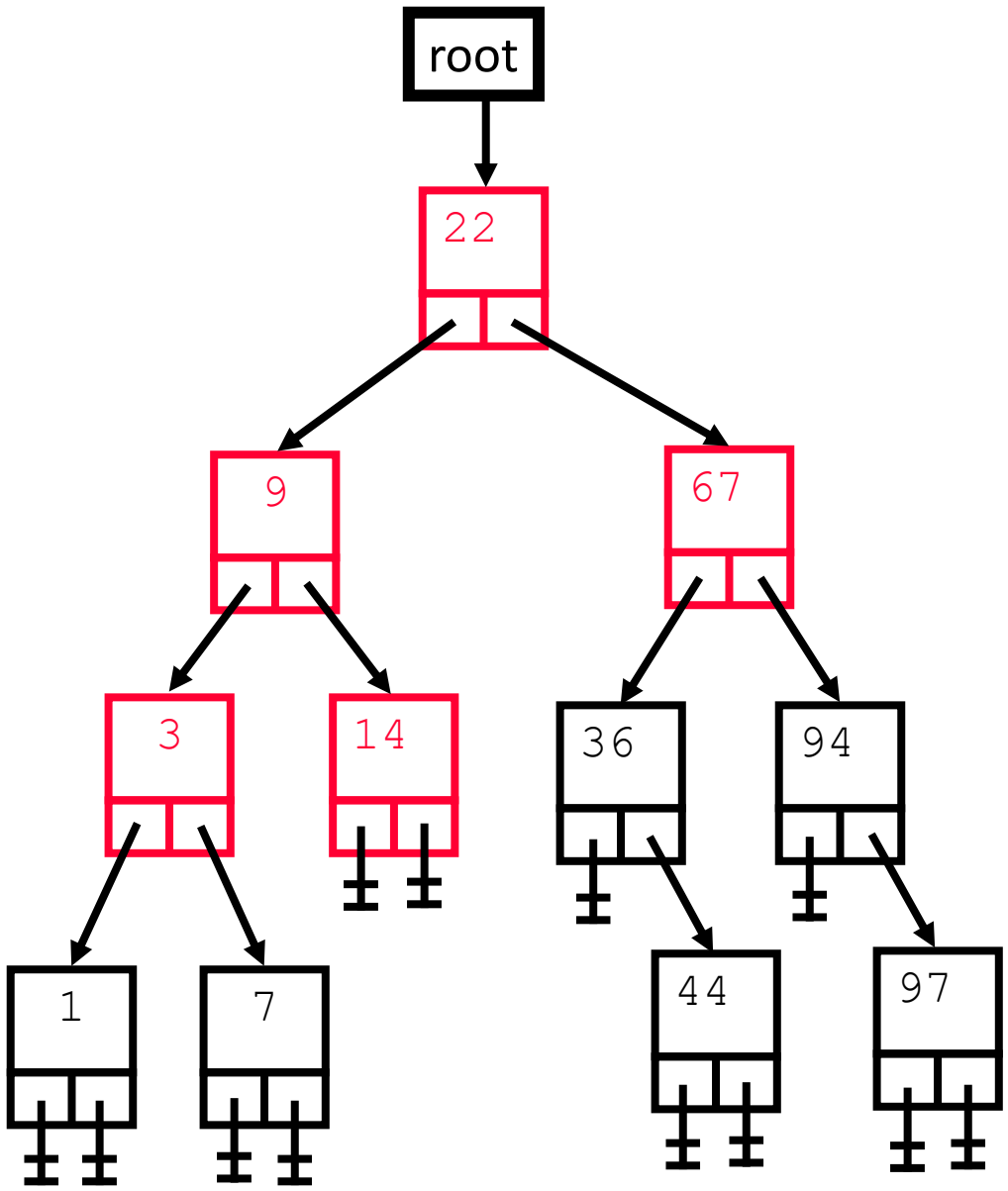
aNode:14



```
Enqueue(root)
loop
  exitif Q empty
  dequeue(aNode)
  enqueue(children)
  print(aNode.data)
endloop
```

Q:36 94 1 7

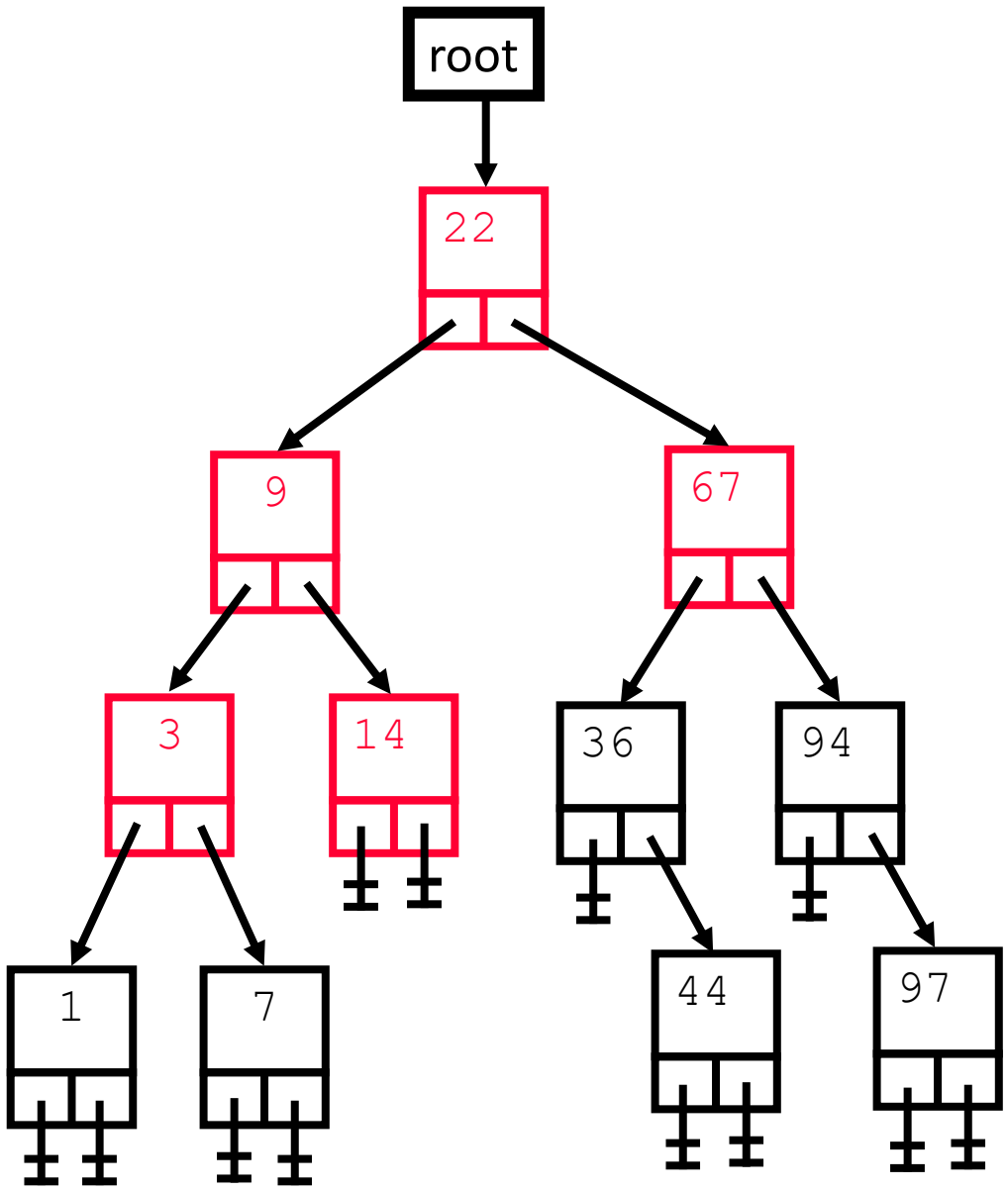
aNode:14



```
Enqueue(root)
loop
  exitif Q empty
  dequeue(aNode)
  enqueue(children)
  print(aNode.data)
endloop
```

Q:36 94 1 7

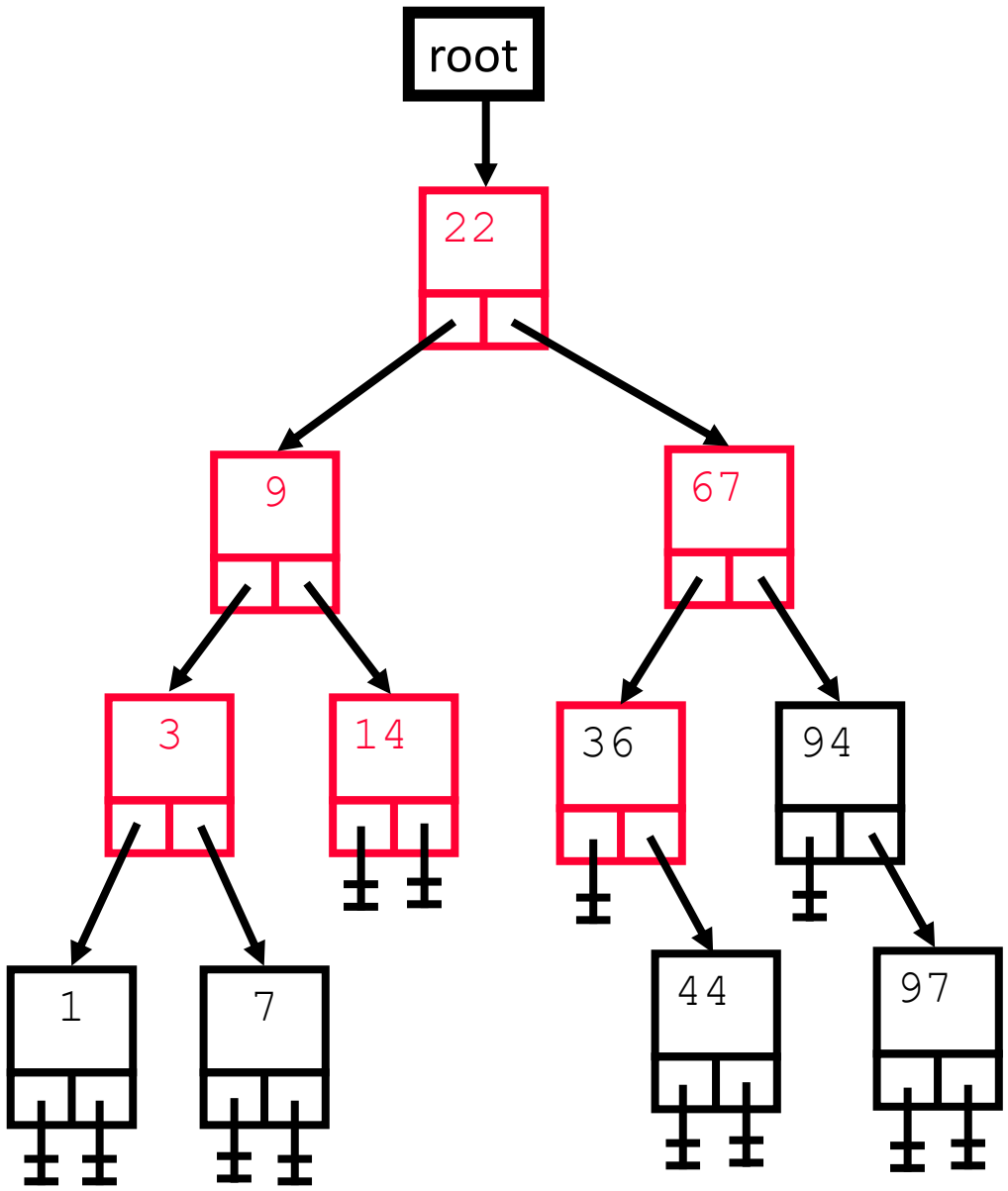
aNode:14



```
Enqueue(root)
loop
  exitif Q empty
  dequeue(aNode)
  enqueue(children)
  print(aNode.data)
endloop
```

Q:94 1 7

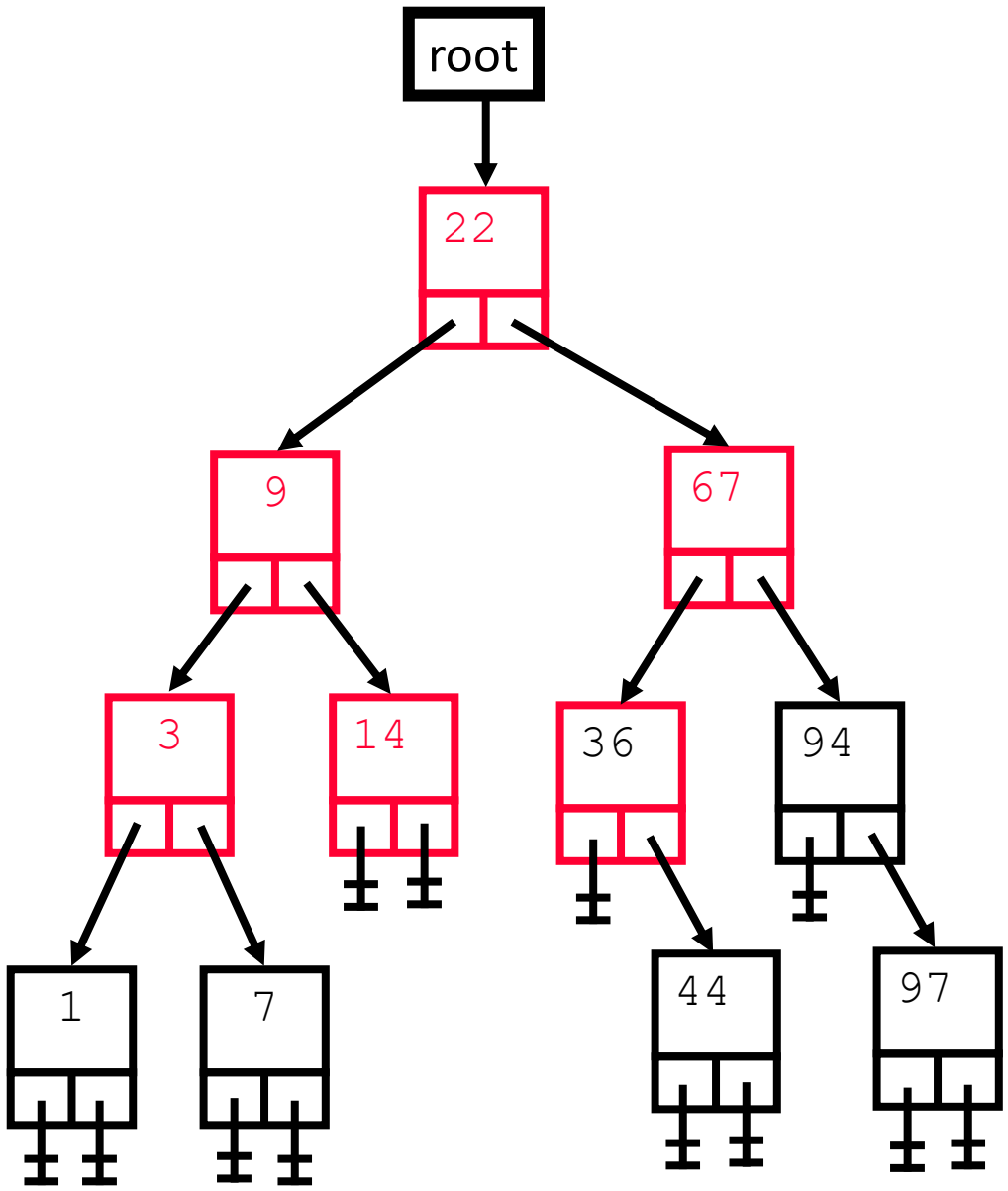
aNode:36



```
Enqueue(root)
loop
  exitif Q empty
  dequeue(aNode)
  enqueue(children)
  print(aNode.data)
endloop
```

Q:94 1 7 44

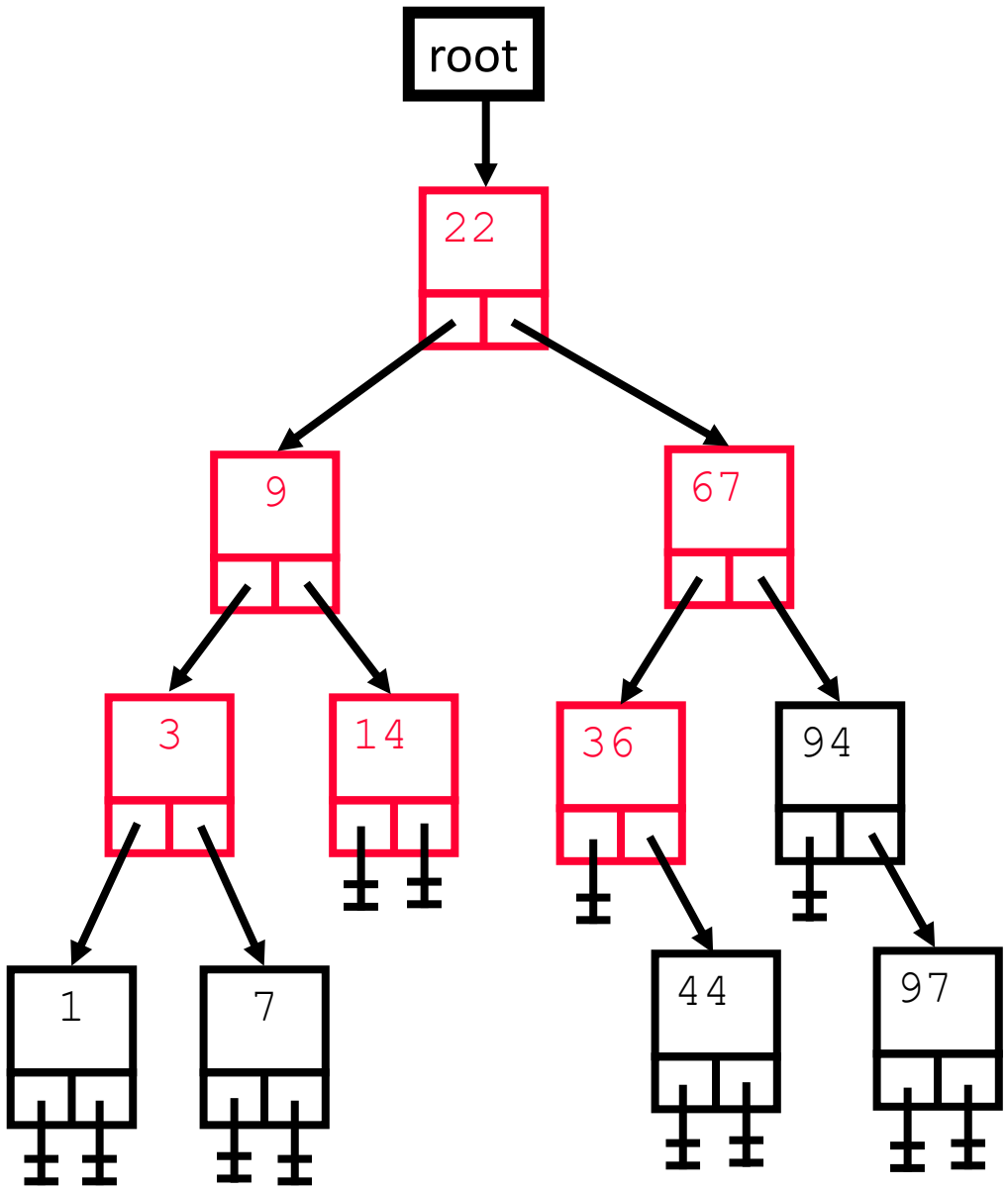
aNode:36



```
Enqueue(root)
loop
  exitif Q empty
  dequeue(aNode)
  enqueue(children)
  print(aNode.data)
endloop
```

Q:94 1 7 44

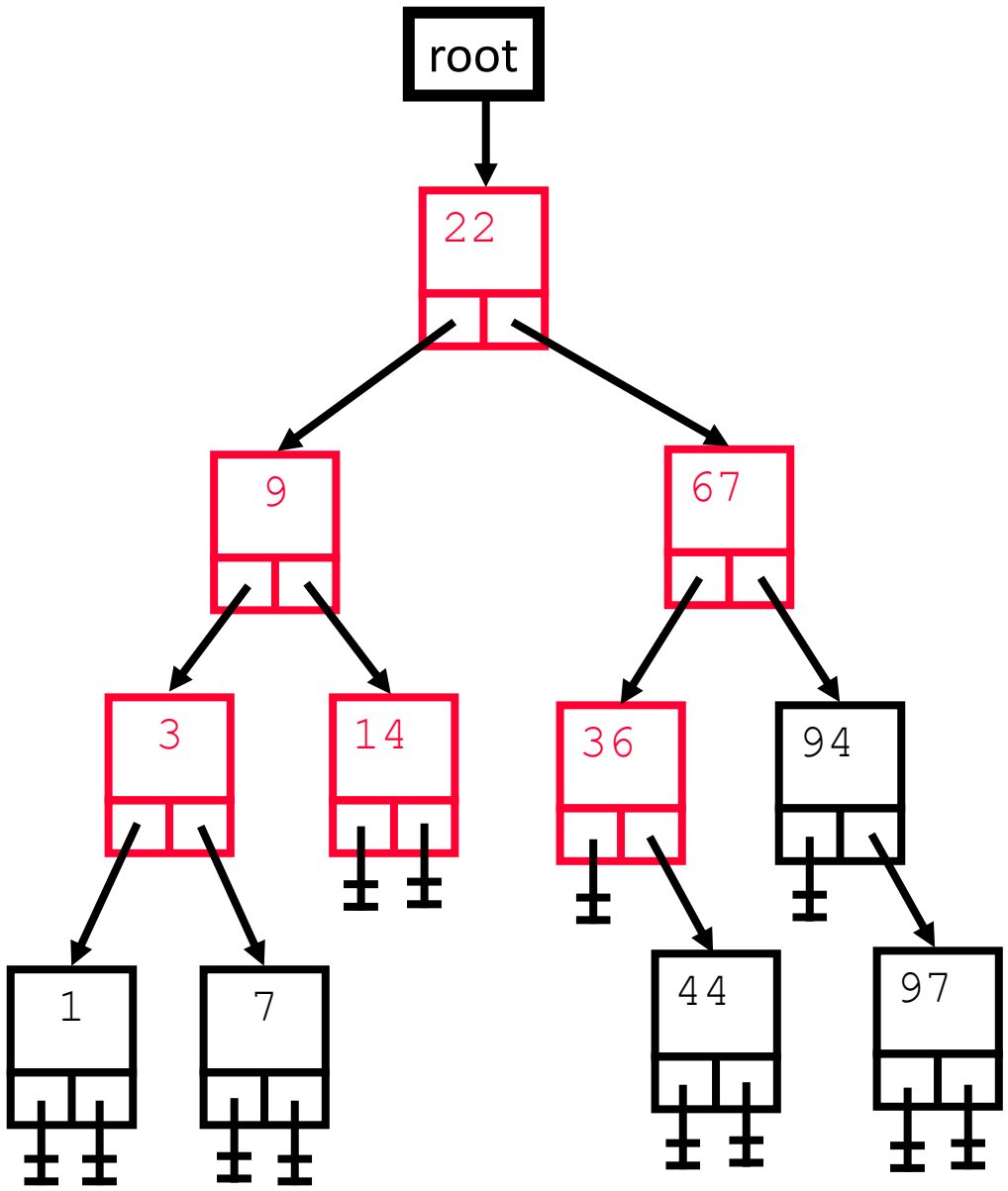
aNode:36



```
Enqueue(root)
loop
  exitif Q empty
  dequeue(aNode)
  enqueue(children)
  print(aNode.data)
endloop
```

Q:94 1 7 44

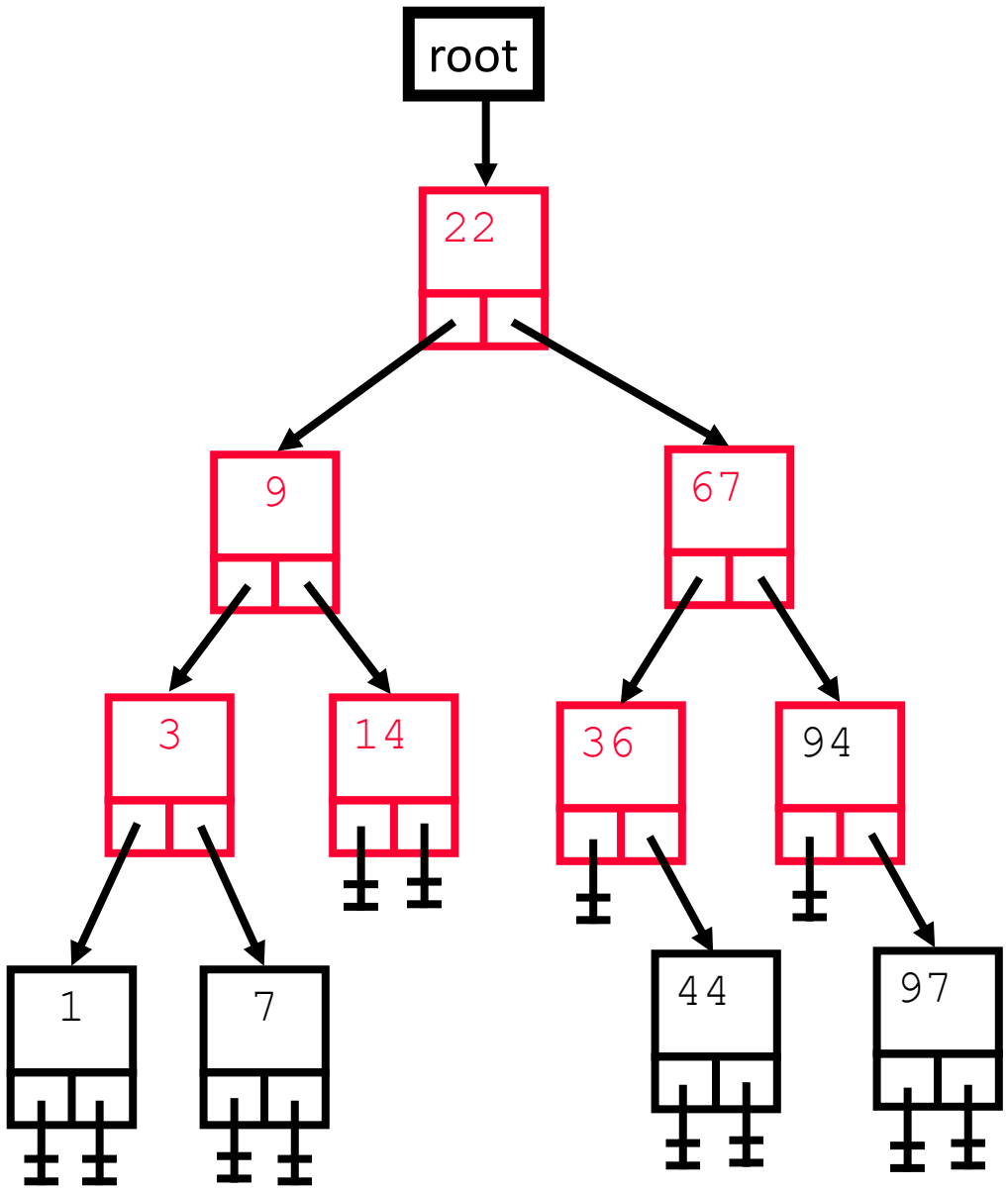
aNode:36



```
Enqueue(root)
loop
  exitif Q empty
  dequeue(aNode)
  enqueue(children)
  print(aNode.data)
endloop
```

Q:1 7 44

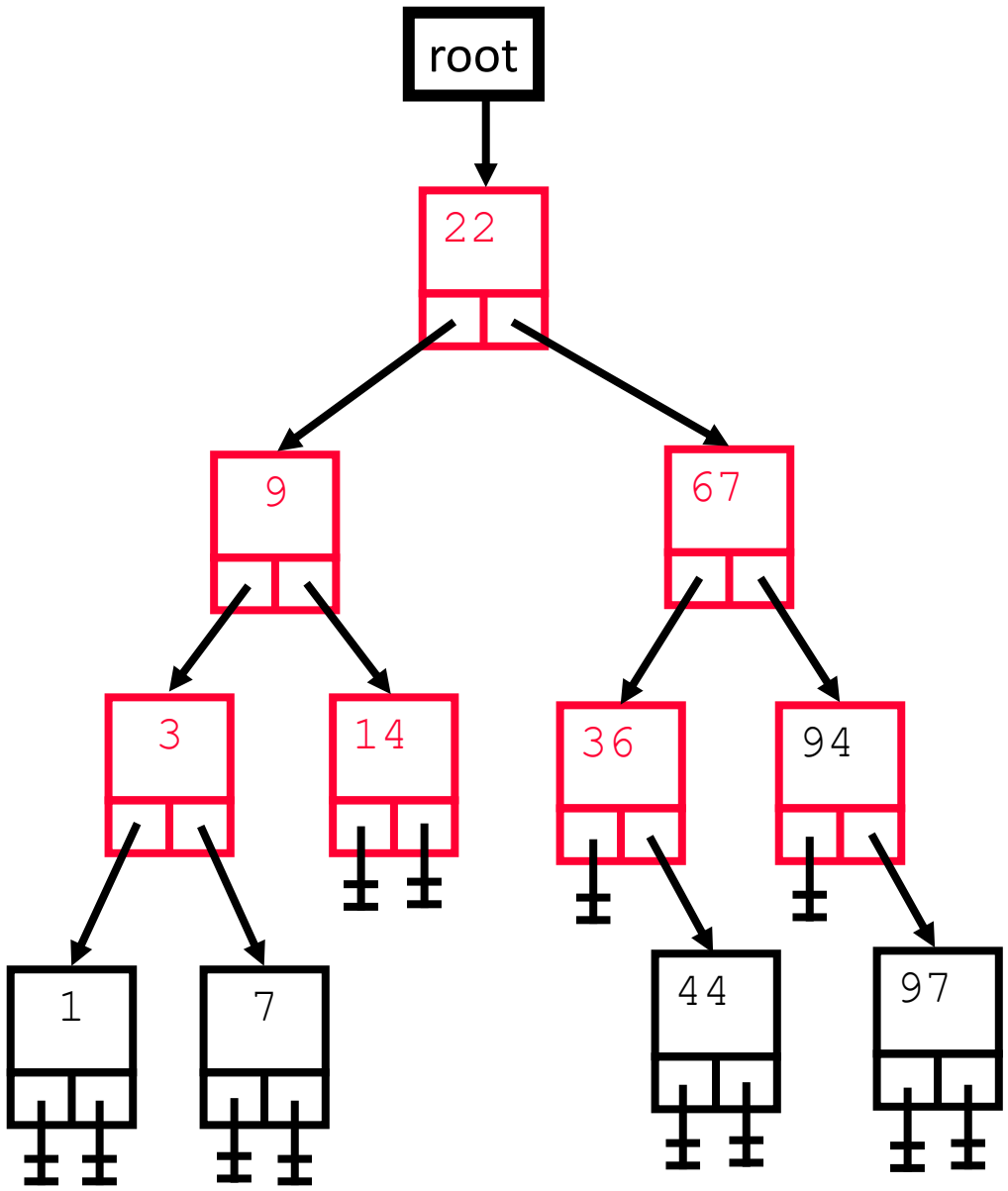
aNode:94



```
Enqueue(root)
loop
  exitif Q empty
  dequeue(aNode)
  enqueue(children)
  print(aNode.data)
endloop
```

Q:1 7 44 97

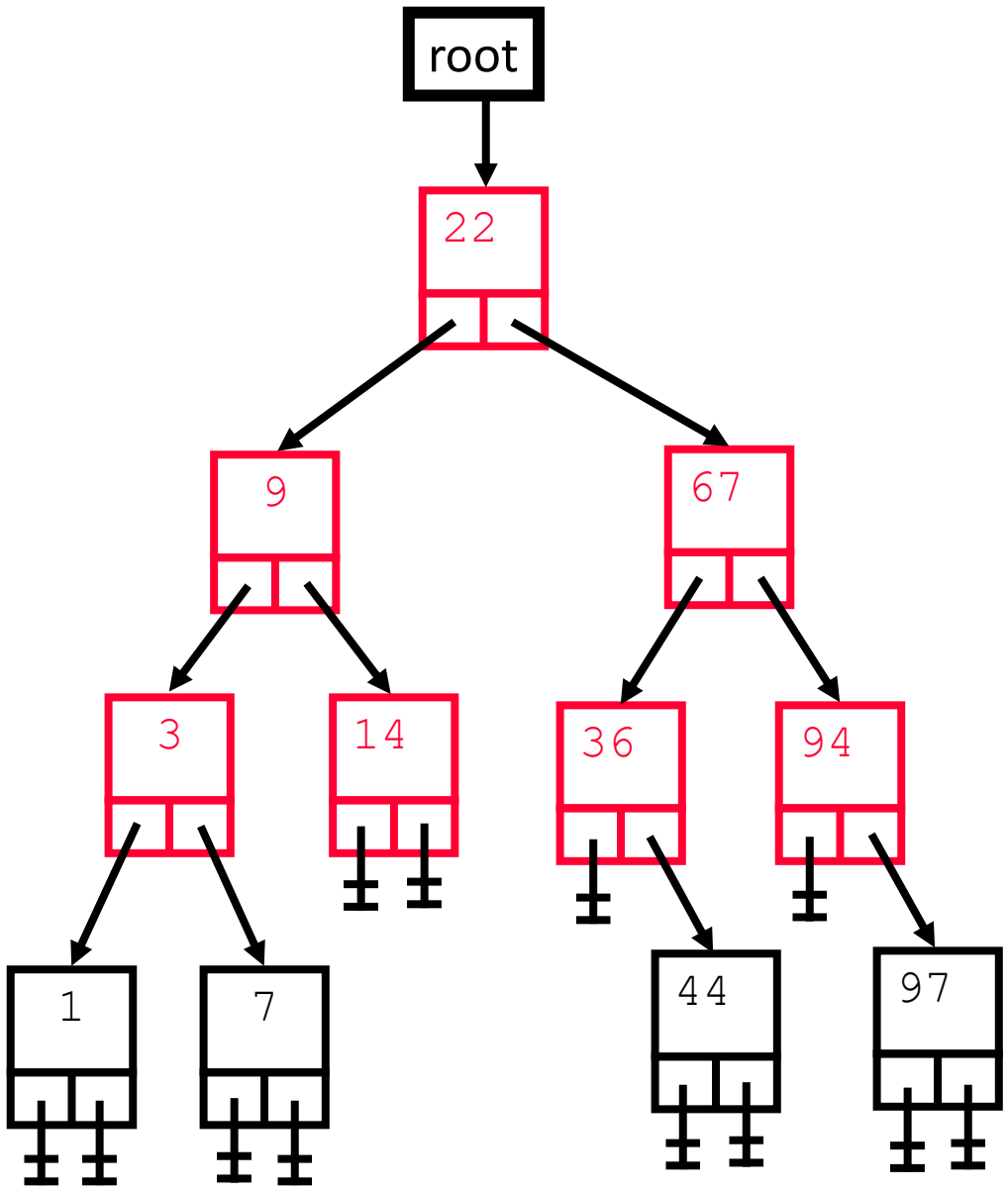
aNode:94



```
Enqueue(root)
loop
  exitif Q empty
  dequeue(aNode)
  enqueue(children)
  print(aNode.data)
endloop
```

Q:1 7 44 97

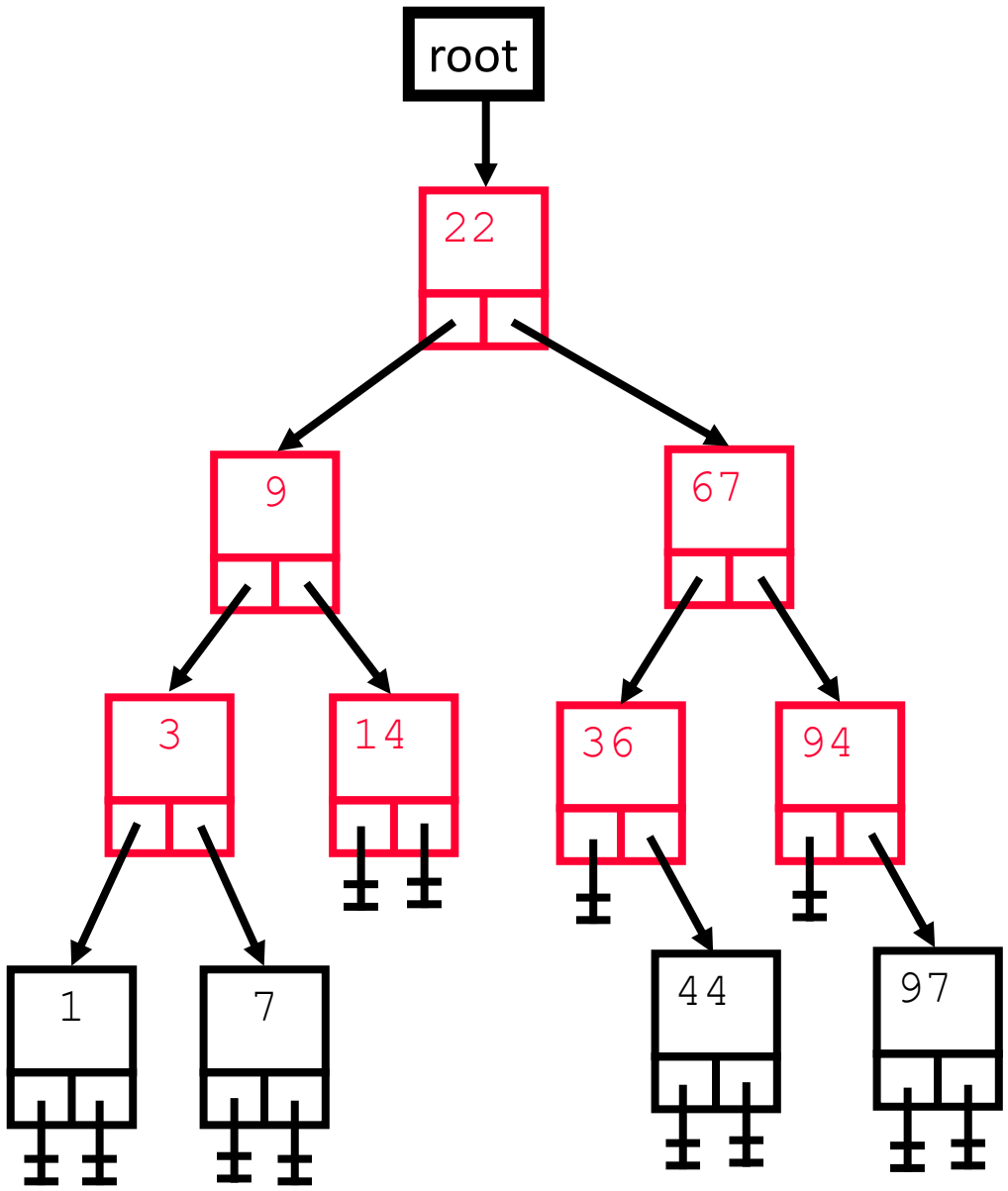
aNode:94



```
Enqueue(root)
loop
  exitif Q empty
  dequeue(aNode)
  enqueue(children)
  print(aNode.data)
endloop
```

Q:1 7 44 97

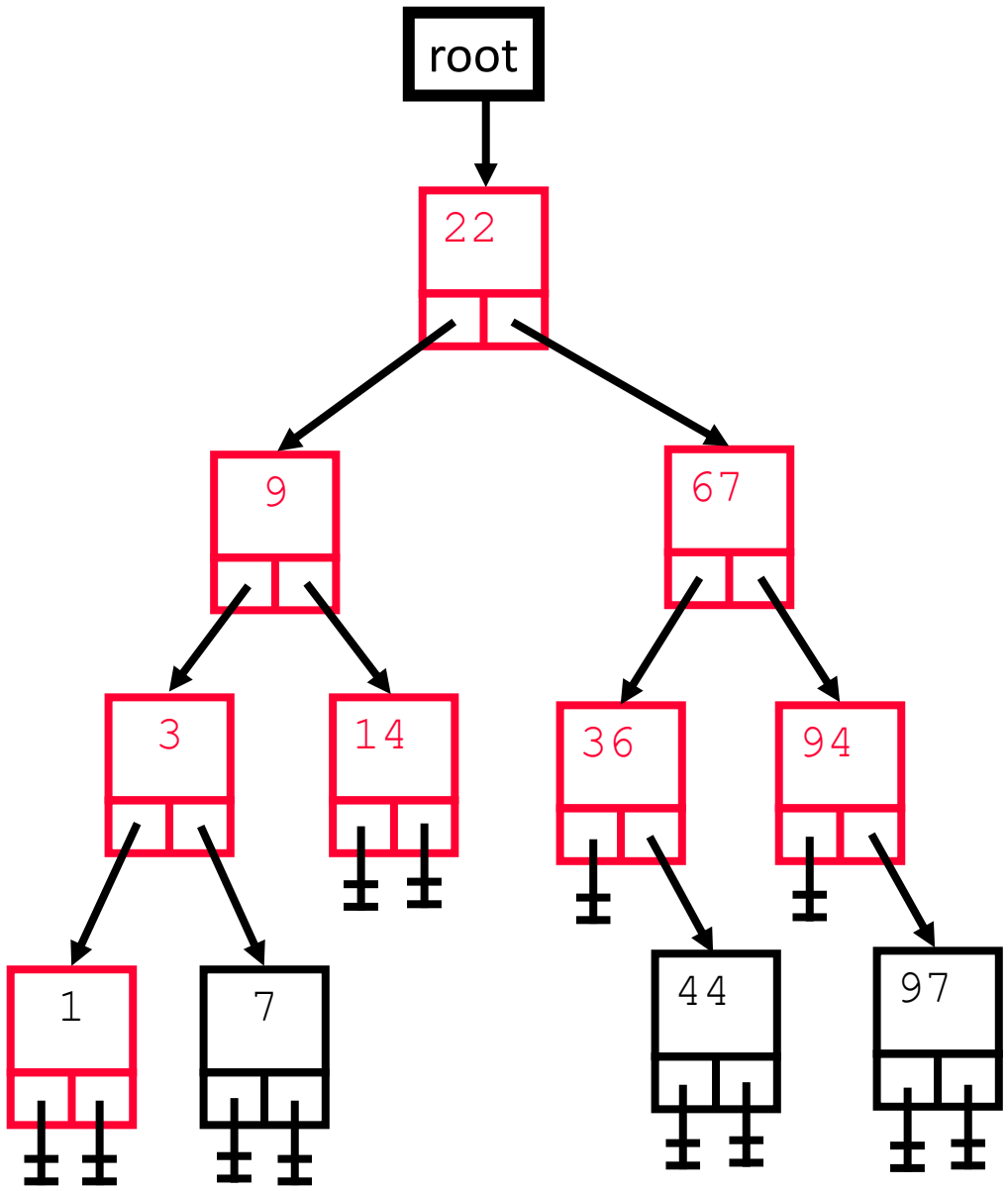
aNode:94



```
Enqueue(root)
loop
  exitif Q empty
  dequeue(aNode)
  enqueue(children)
  print(aNode.data)
endloop
```

Q:7 44 97

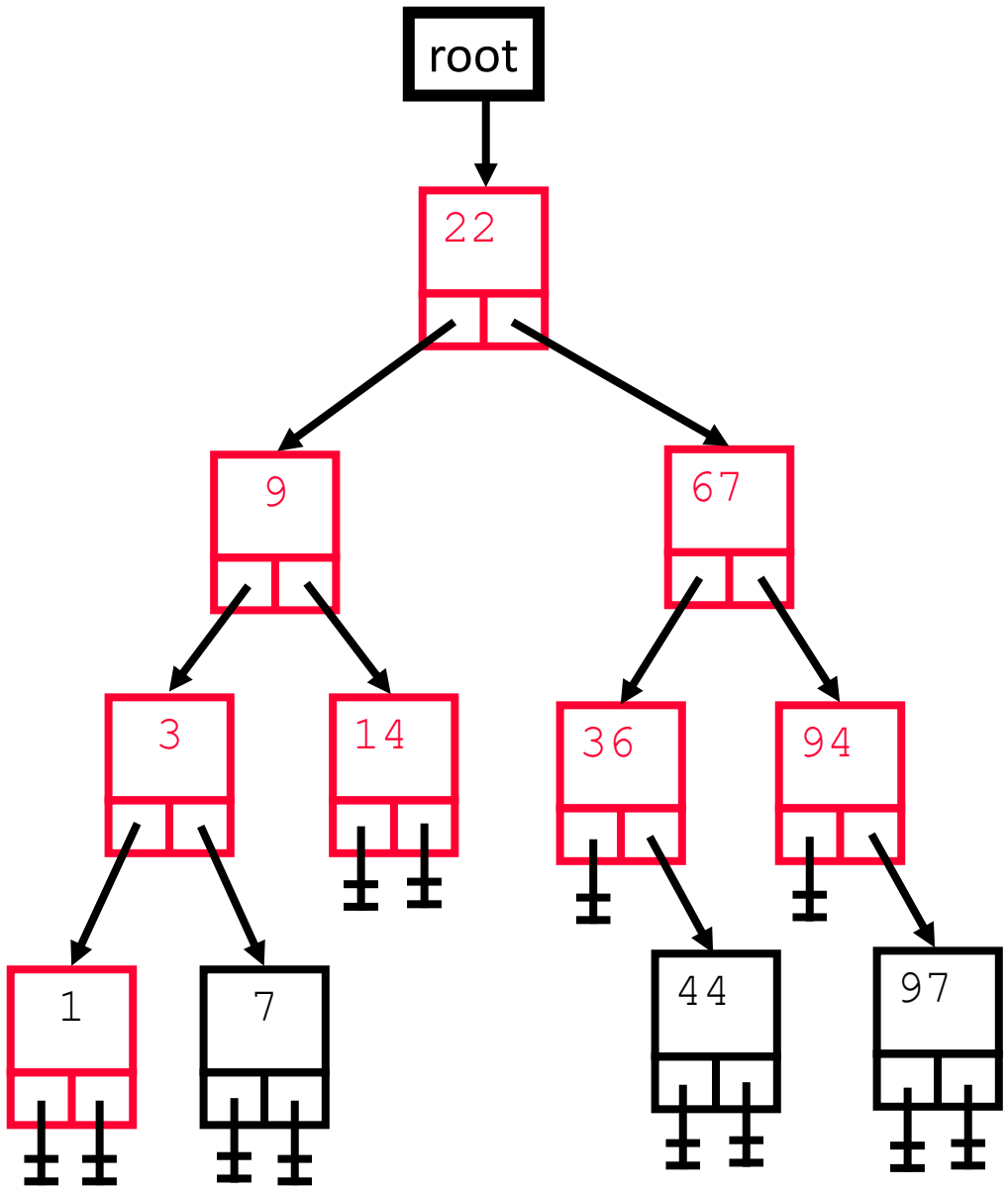
aNode:1



```
Enqueue(root)
loop
  exitif Q empty
  dequeue(aNode)
  enqueue(children)
  print(aNode.data)
endloop
```

Q:7 44 97

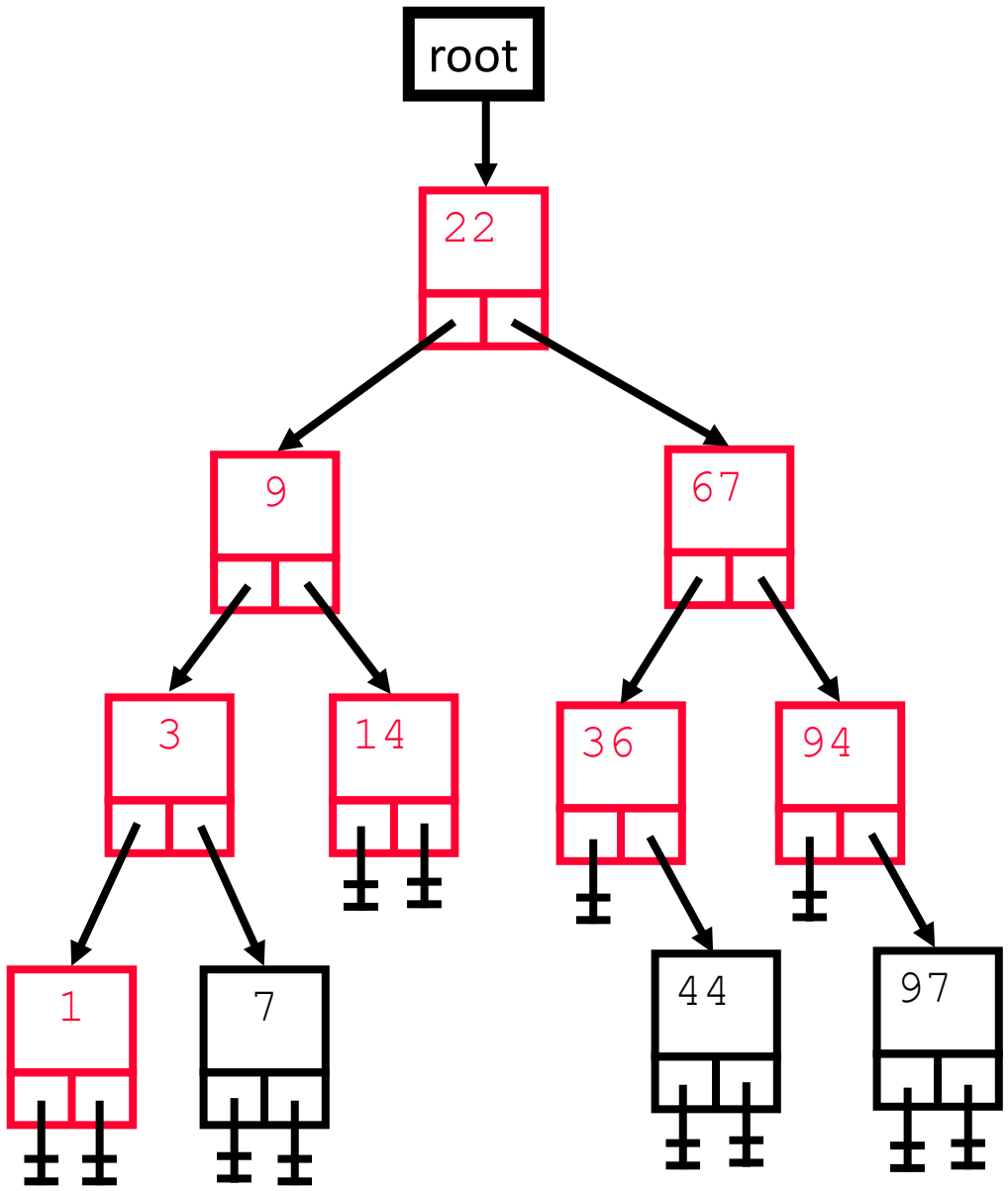
aNode:1



```
Enqueue(root)
loop
  exitif Q empty
  dequeue(aNode)
  enqueue(children)
  print(aNode.data)
endloop
```

Q:7 44 97

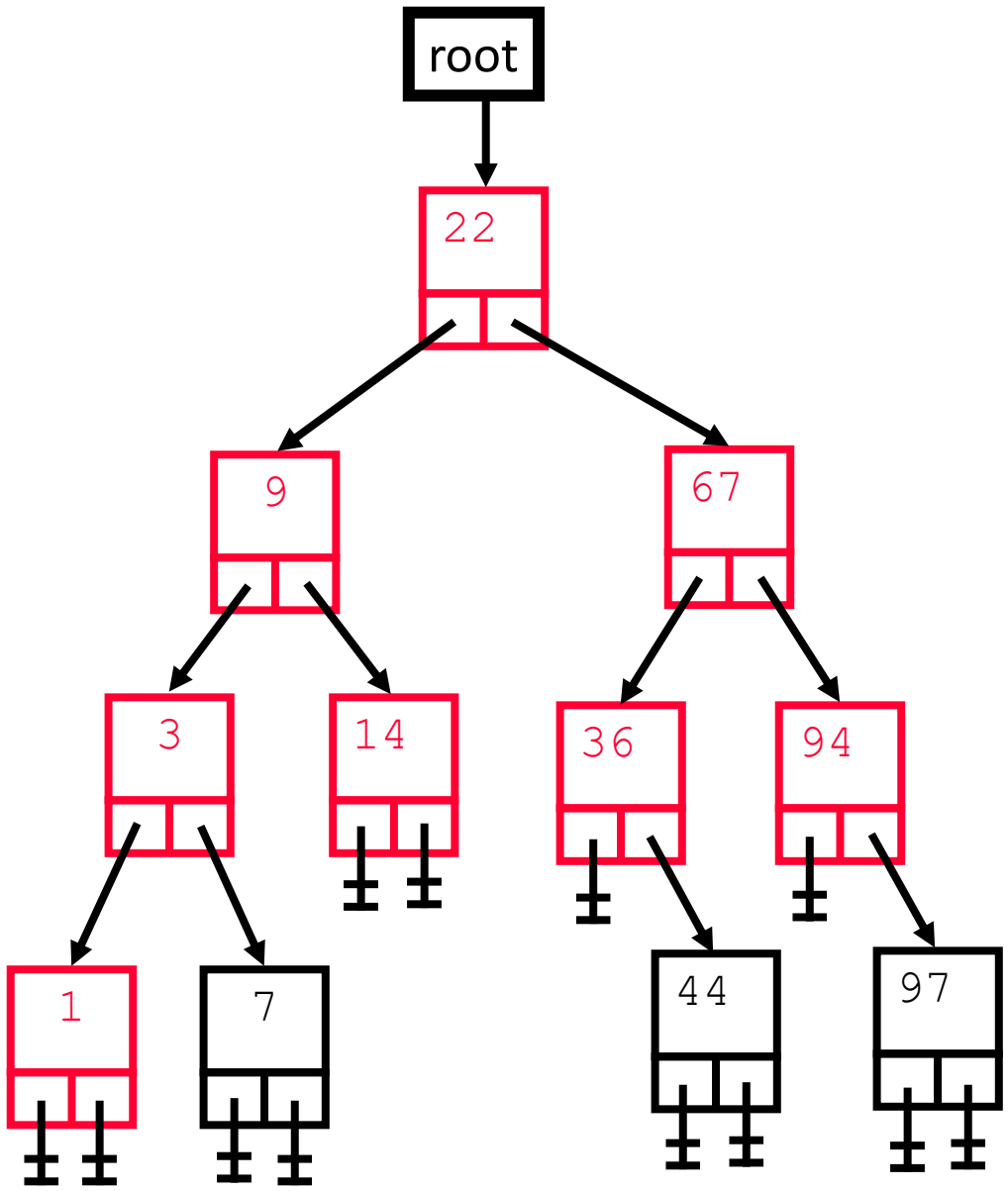
aNode:1



```
Enqueue(root)
loop
  exitif Q empty
  dequeue(aNode)
  enqueue(children)
  print(aNode.data)
endloop
```

Q:7 44 97

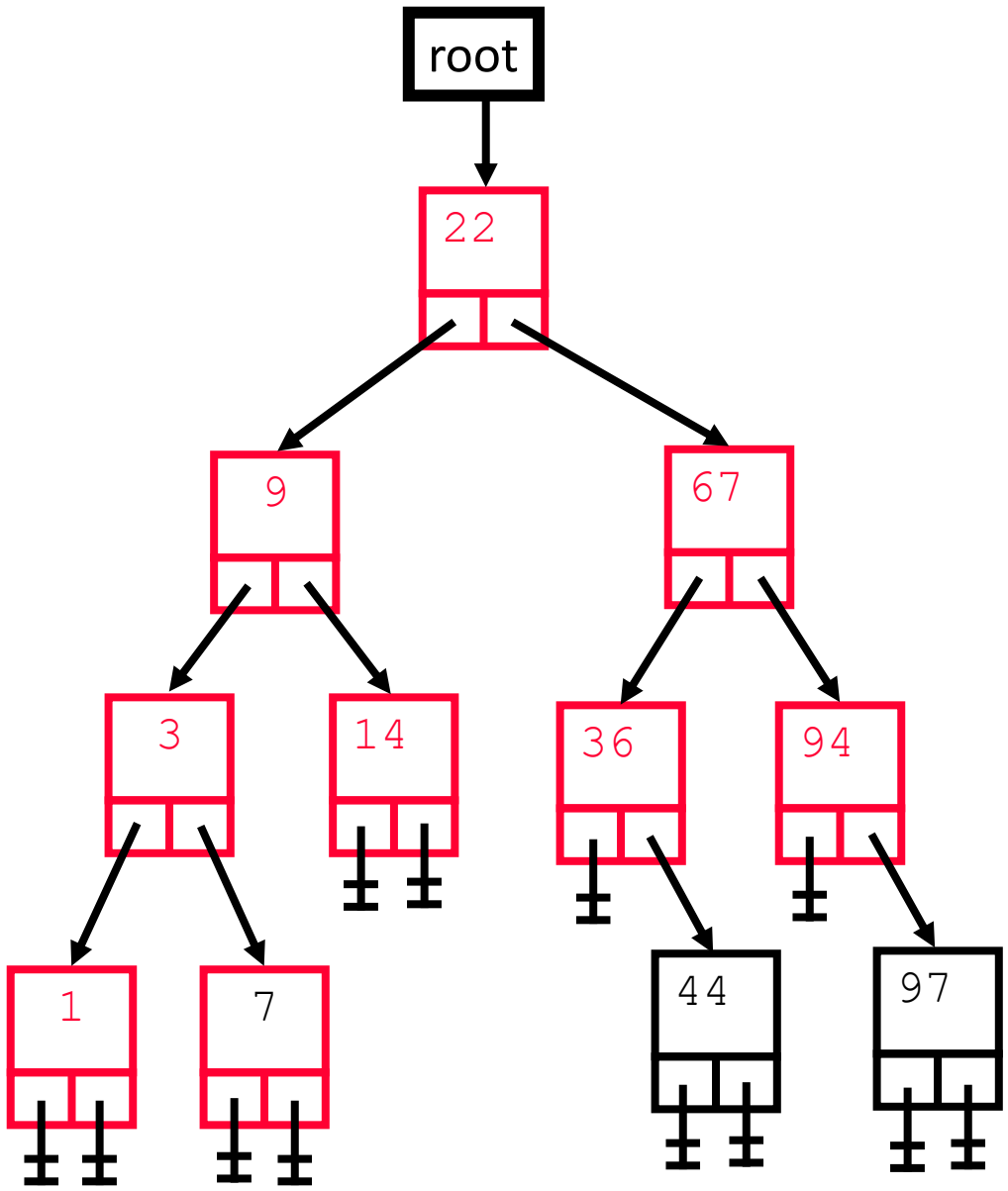
aNode:1



```
Enqueue(root)
loop
  exitif Q empty
  dequeue(aNode)
  enqueue(children)
  print(aNode.data)
endloop
```

Q:44 97

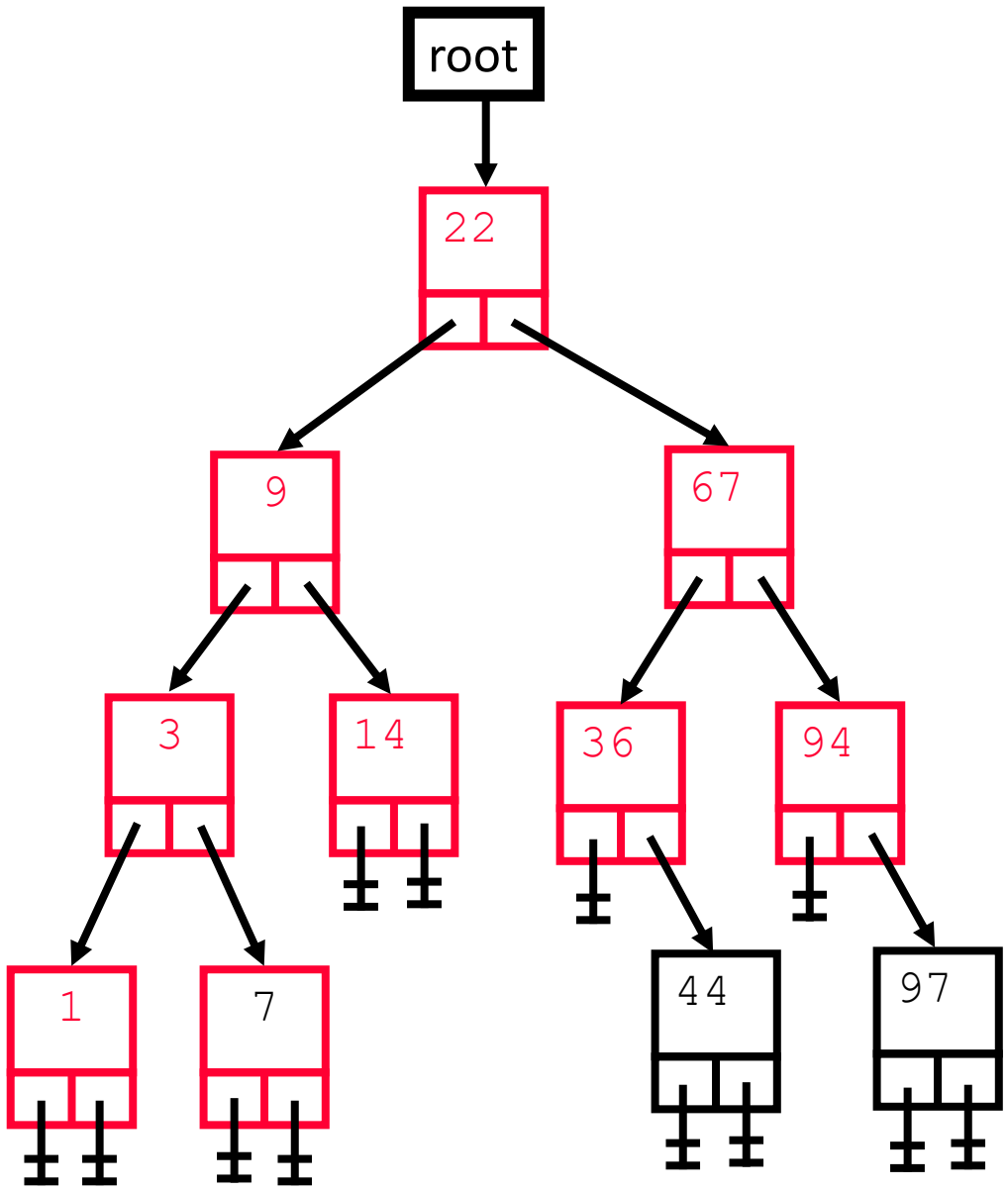
aNode:7



```
Enqueue(root)
loop
  exitif Q empty
  dequeue(aNode)
  enqueue(children)
  print(aNode.data)
endloop
```

Q:44 97

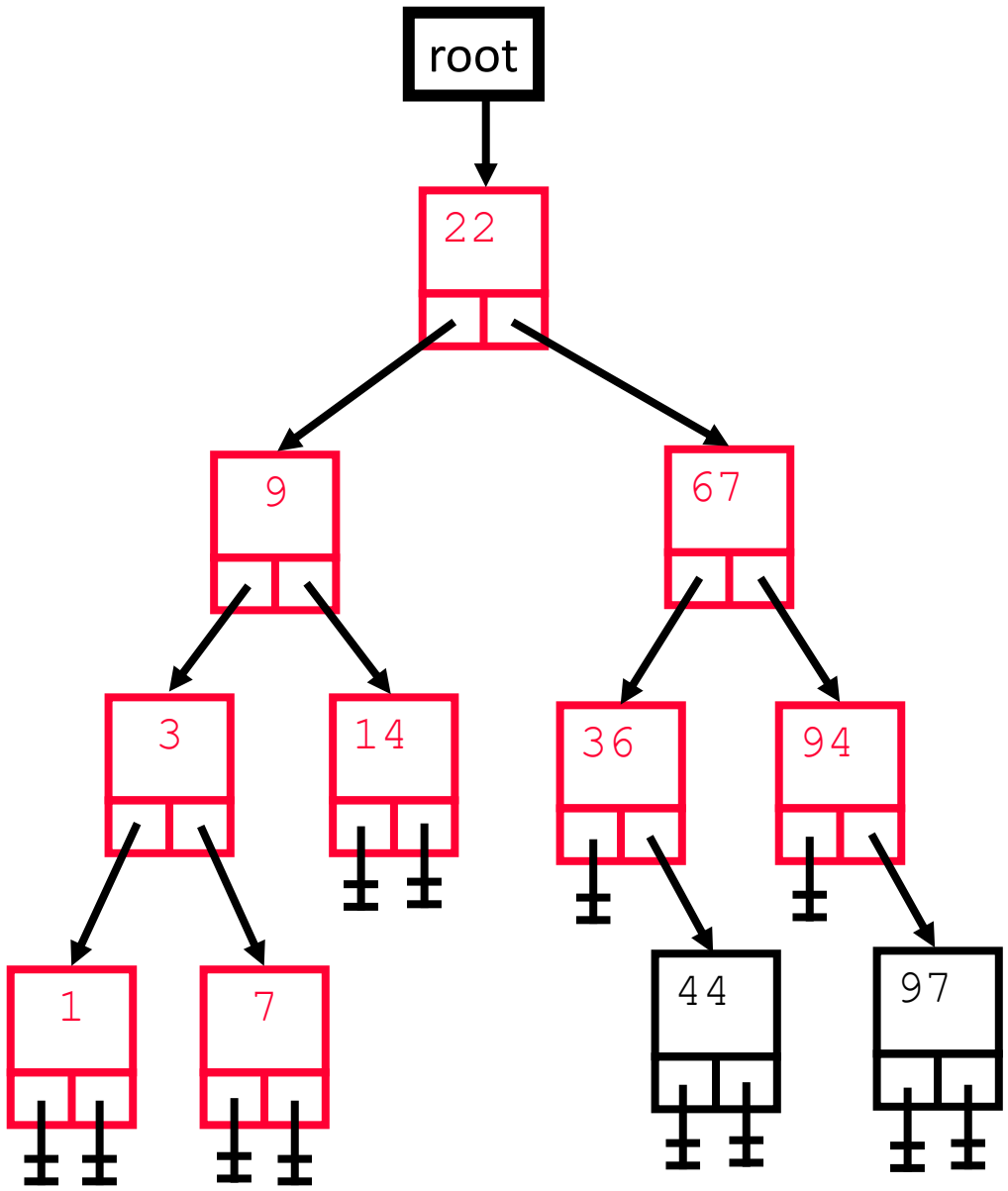
aNode:7



```
Enqueue(root)
loop
  exitif Q empty
  dequeue(aNode)
  enqueue(children)
  print(aNode.data)
endloop
```

Q:44 97

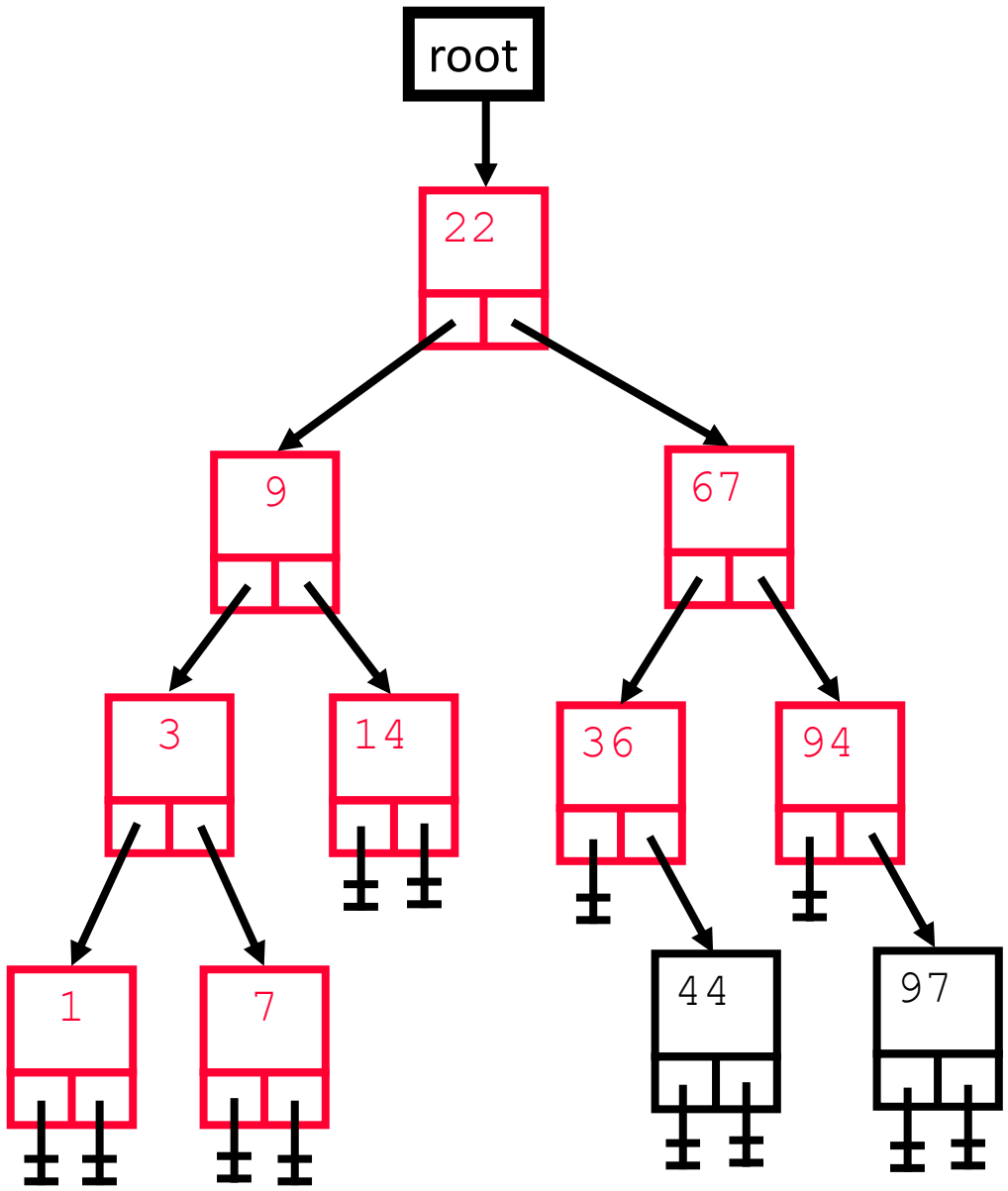
aNode:7



```
Enqueue(root)
loop
  exitif Q empty
  dequeue(aNode)
  enqueue(children)
  print(aNode.data)
endloop
```

Q:44 97

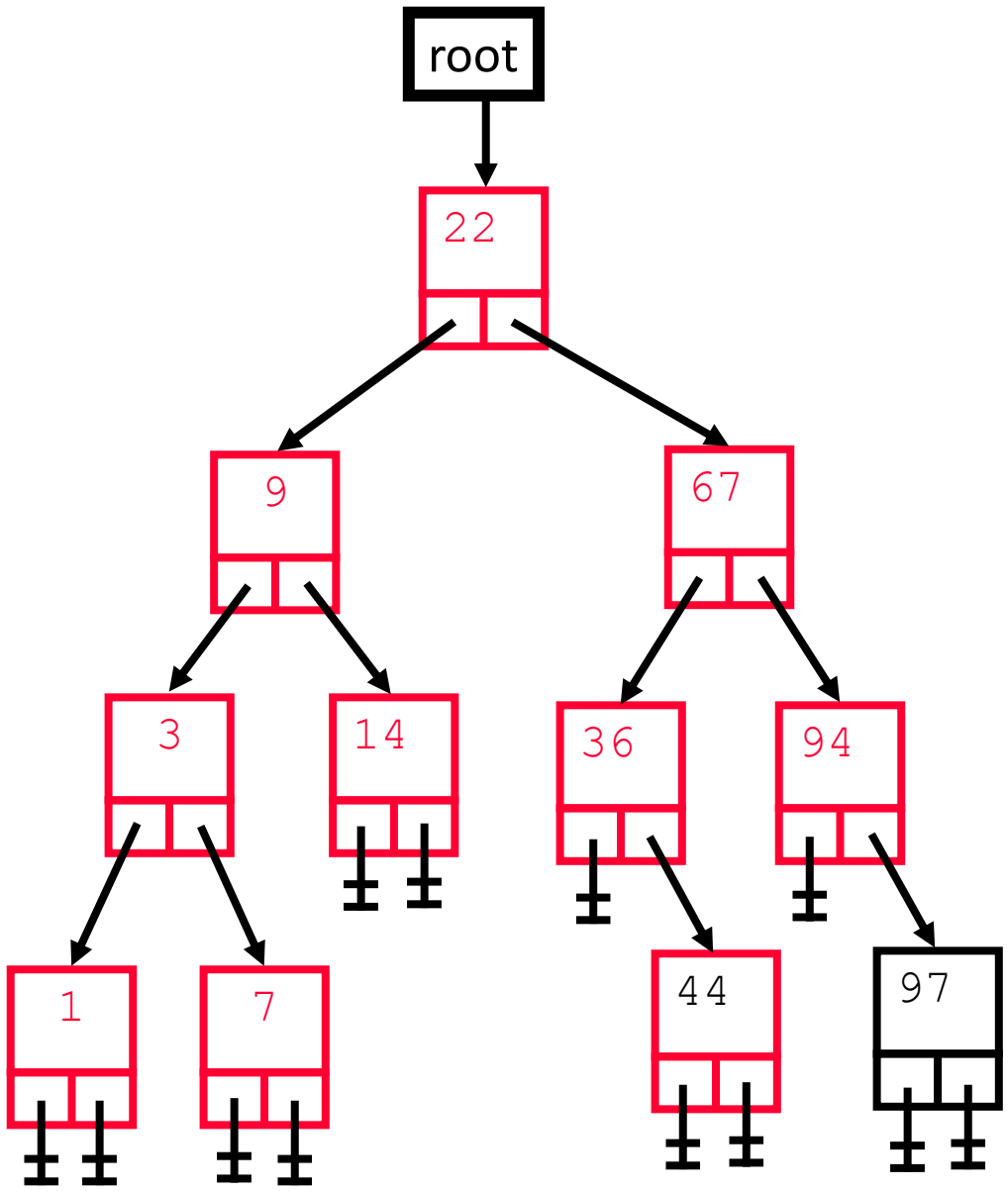
aNode:7



```
Enqueue(root)
loop
  exitif Q empty
  dequeue(aNode)
  enqueue(children)
  print(aNode.data)
endloop
```

Q:97

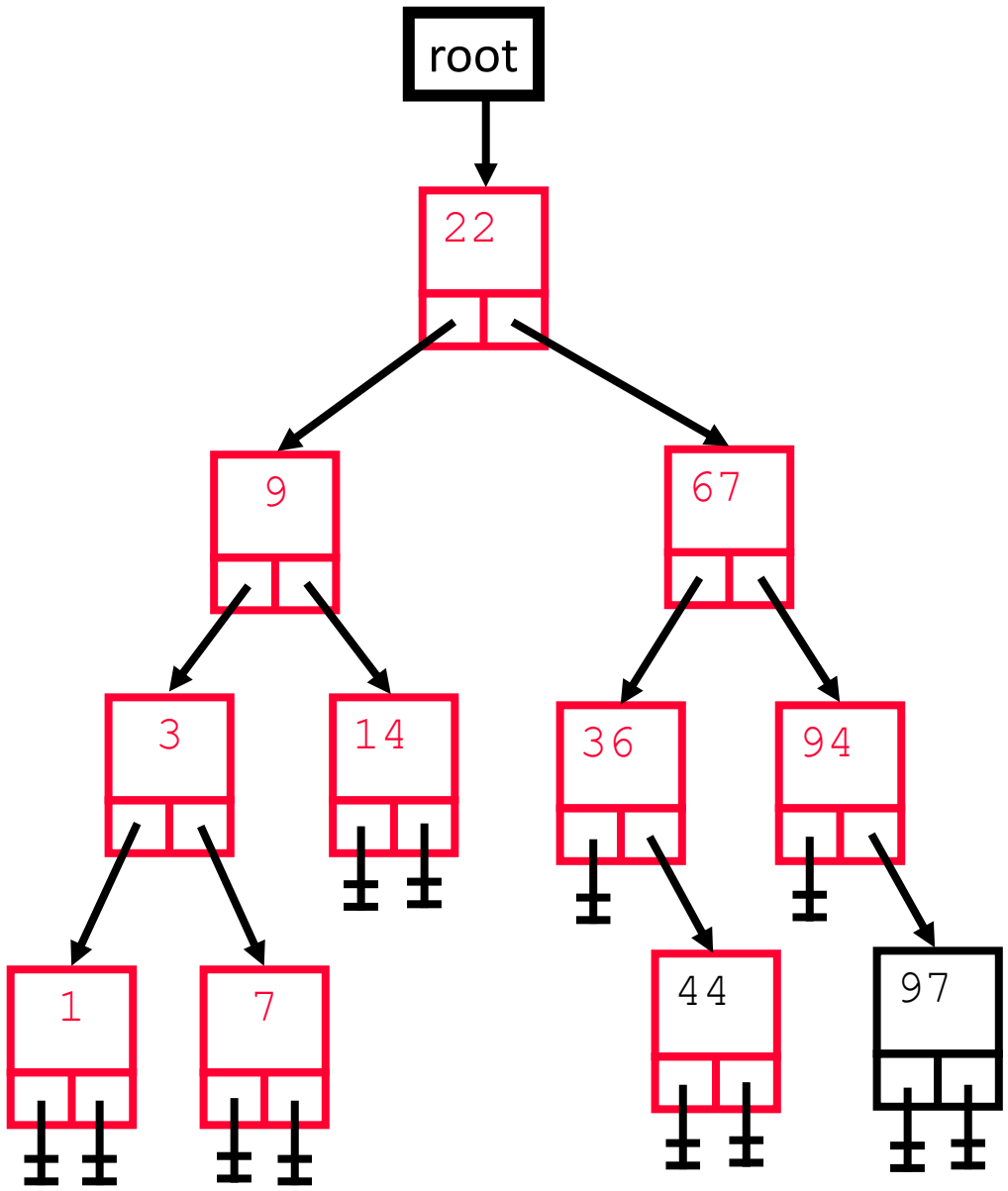
aNode:44



```
Enqueue(root)
loop
  exitif Q empty
  dequeue(aNode)
  enqueue(children)
  print(aNode.data)
endloop
```

Q:97

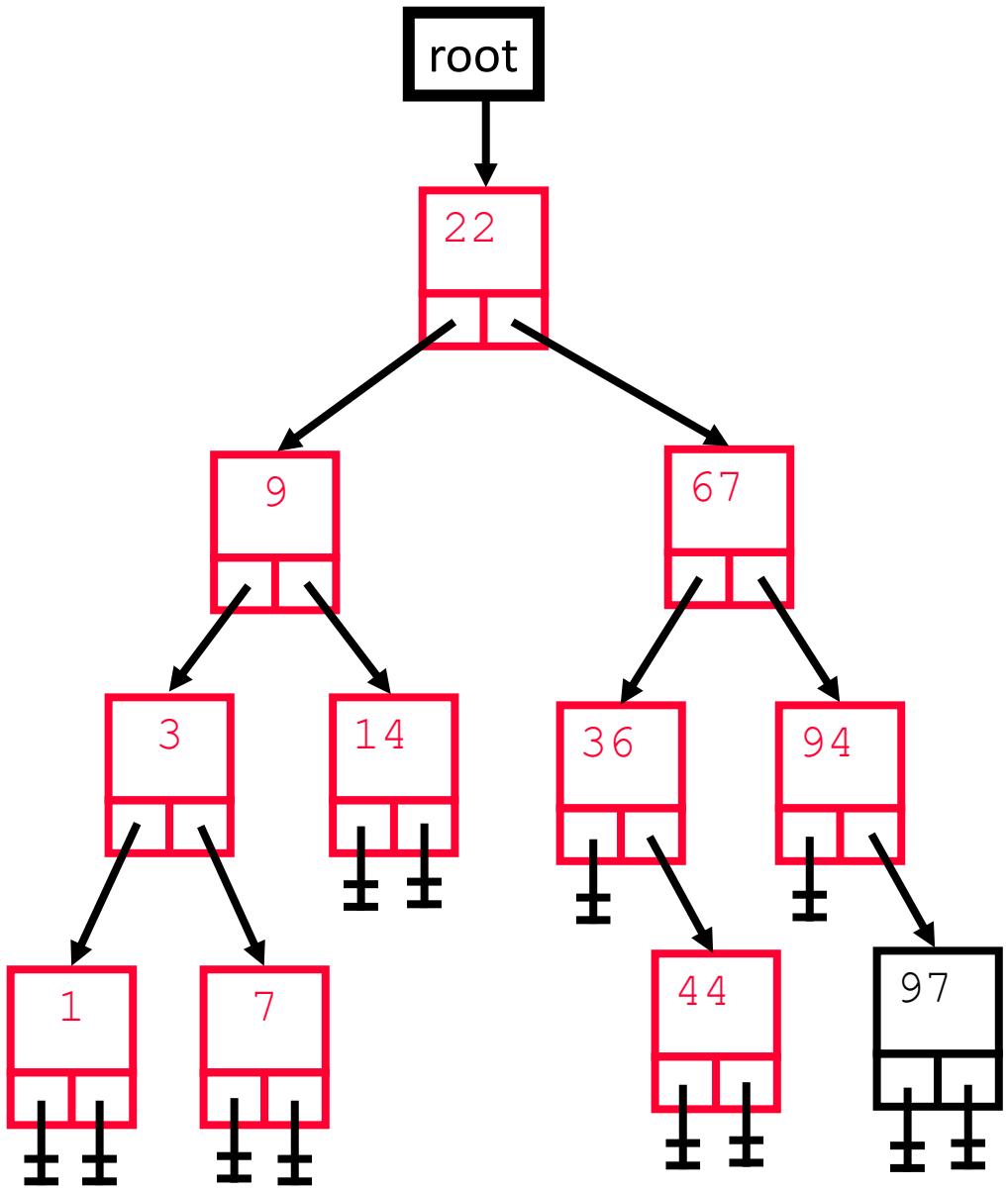
aNode:44



```
Enqueue(root)
loop
  exitif Q empty
  dequeue(aNode)
  enqueue(children)
  print(aNode.data)
endloop
```

Q:97

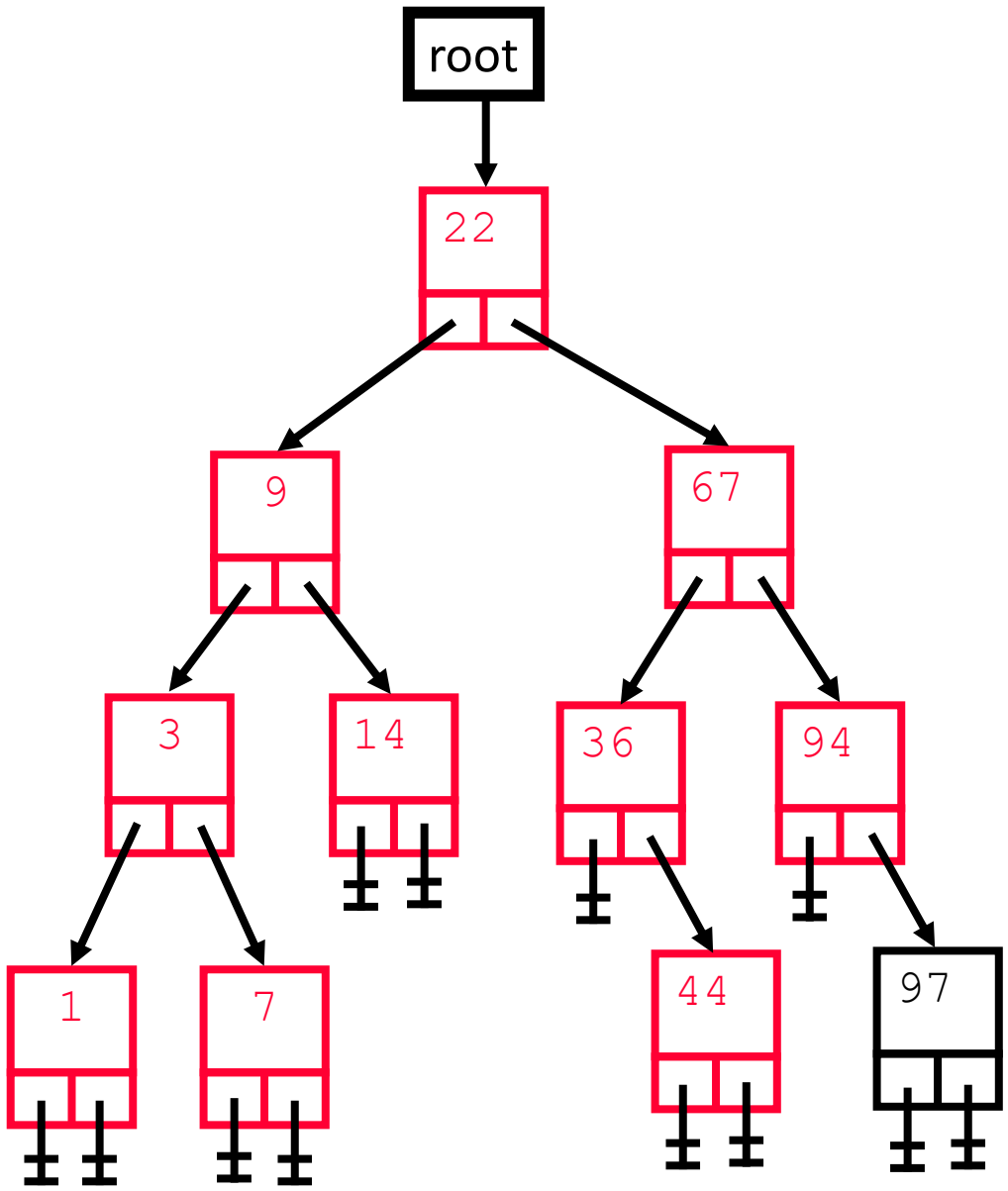
aNode:44



```
Enqueue(root)
loop
  exitif Q empty
  dequeue(aNode)
  enqueue(children)
  print(aNode.data)
endloop
```

Q:97

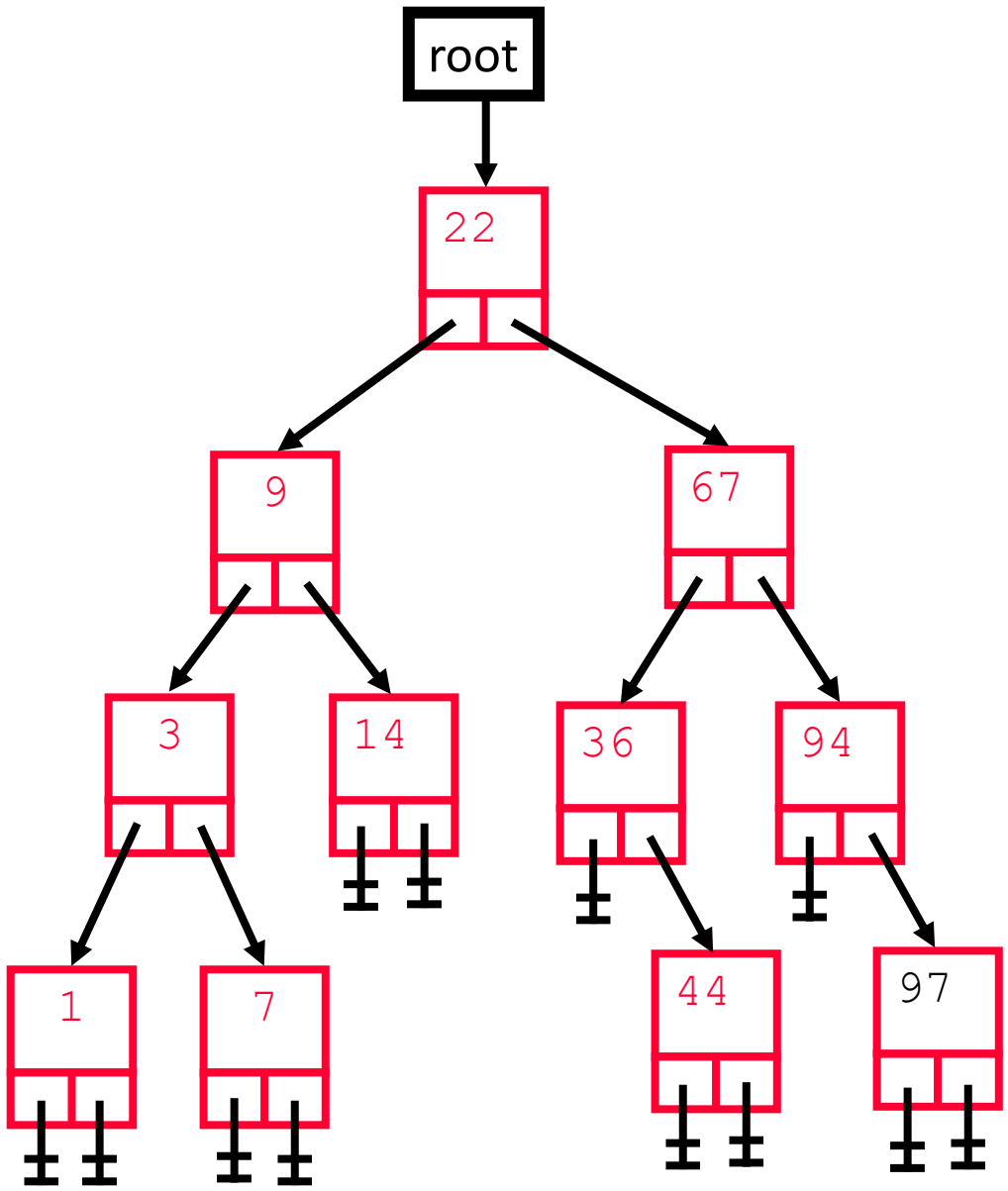
aNode:44



```
Enqueue(root)
loop
  exitif Q empty
  dequeue(aNode)
  enqueue(children)
  print(aNode.data)
endloop
```

Q:

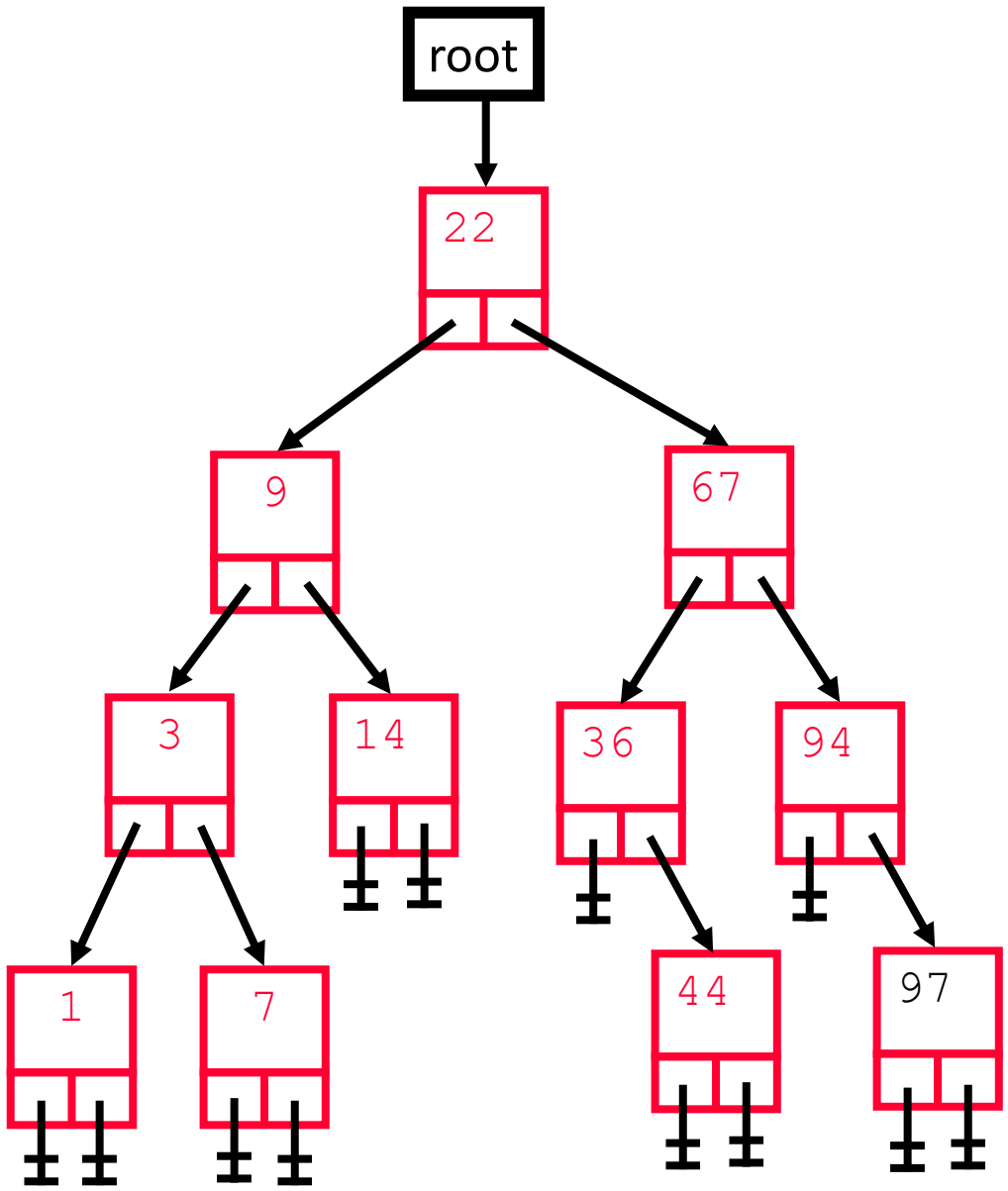
aNode:97



```
Enqueue(root)
loop
  exitif Q empty
  dequeue(aNode)
  enqueue(children)
  print(aNode.data)
endloop
```

Q:

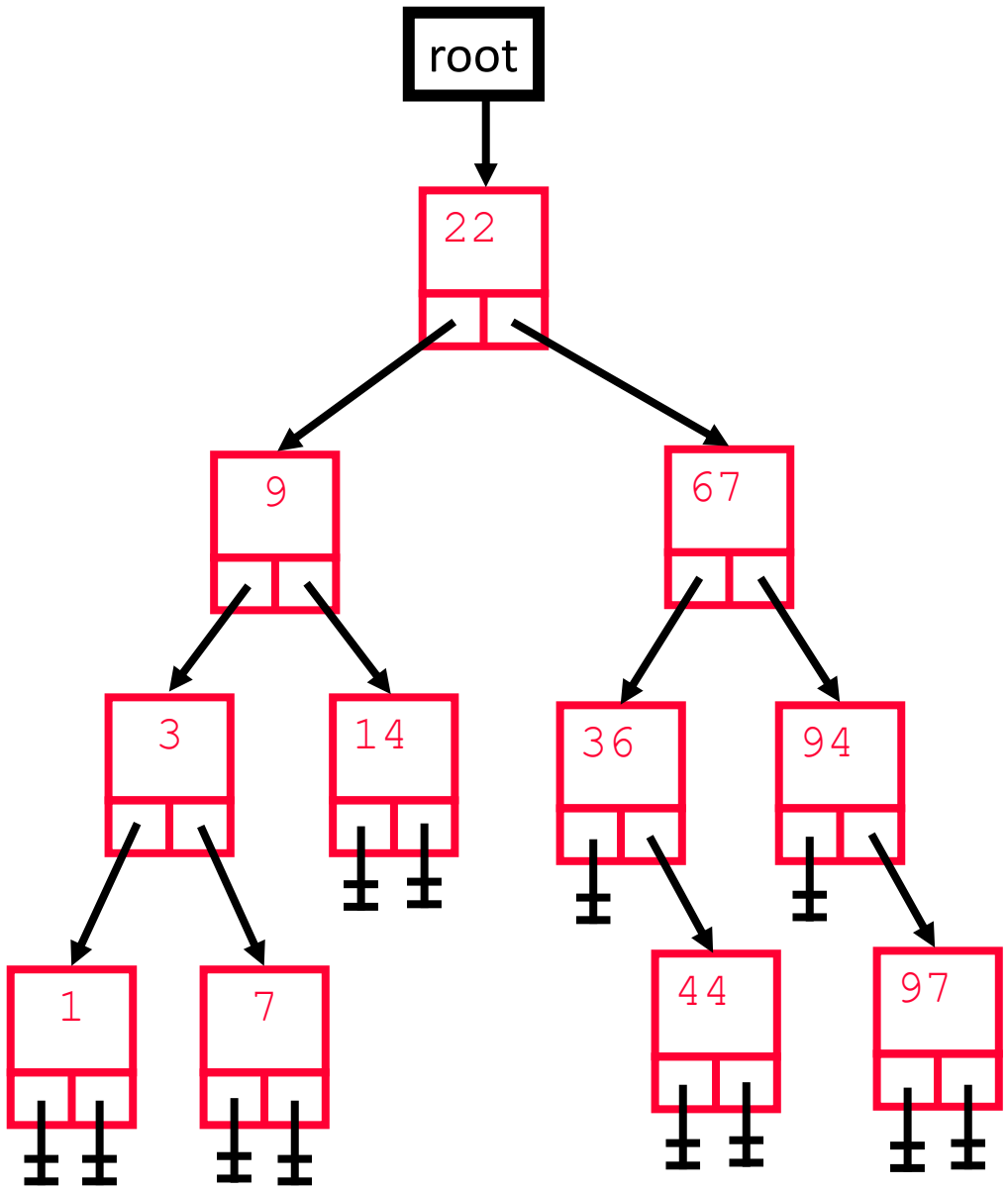
aNode:97



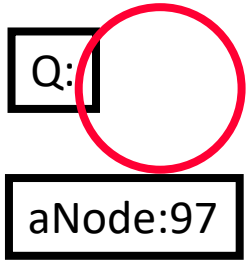
```
Enqueue(root)
loop
  exitif Q empty
  dequeue(aNode)
  enqueue(children)
  print(aNode.data)
endloop
```

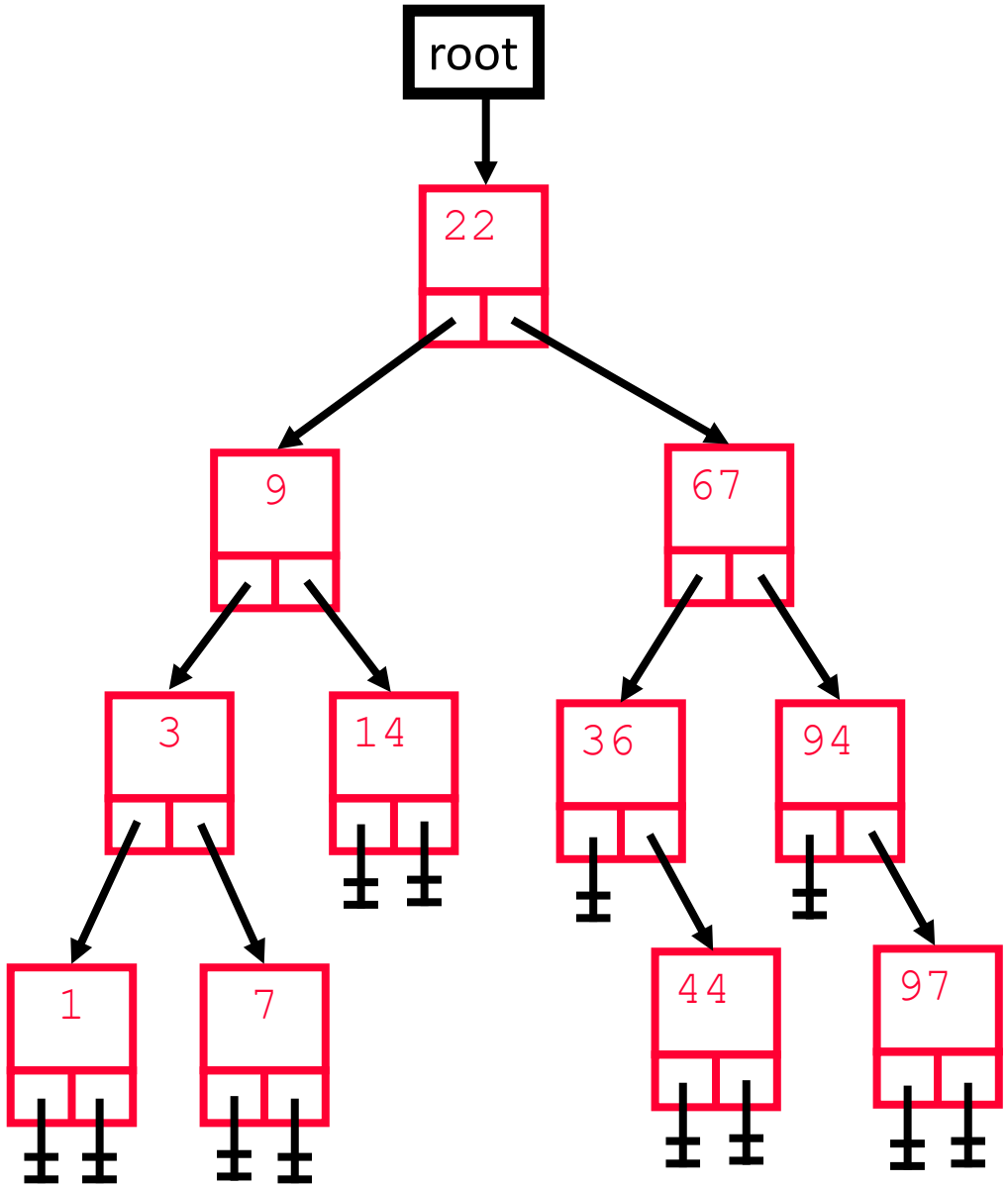
Q:

aNode:97



```
Enqueue(root)
loop
  exitif Q empty
  dequeue(aNode)
  enqueue(children)
  print(aNode.data)
endloop
```





Reference

- https://www.cc.gatech.edu/~bleahy/
- **Prof. Bill Leahy, Georgia Tech. College of Computing**

Thank You