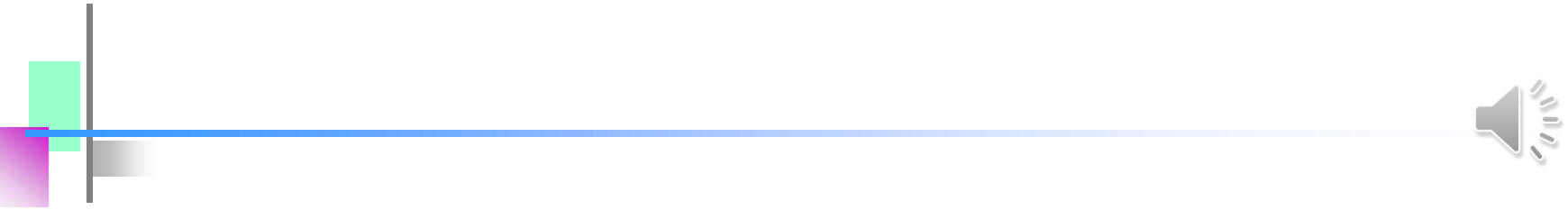


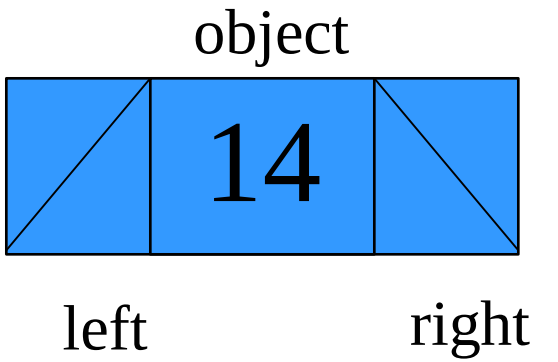
BST Implementation

C++ Implementation

- We need four files:
  - TreeNode.h // class that will create Nodes
  - Tree.h /* class that will contain all functions e.g insert, update, delete etc */
  - Tree.cpp // function definition file of class Tree
  - Main.cpp /* contain main menu, object creation and function calling*/

C++ Implementation (TreeNode.h)

```
class TreeNode {
private:
        int   object;
        TreeNode* left;
        TreeNode* right;
public:

         // constructor
         TreeNode()
         {
                 this->object = 0;
                 this->left = this->right = NULL;
         };
```



C++ Implementation

```
int getInfo()
{
  return this->object;
};
void setInfo(int info)
{
  this->object = info;
};
TreeNode* getLeft()
{
  return left;
};
void setLeft(TreeNode *left)
{
  this->left = left;
};
```

C++ Implementation

```
TreeNode *getRight()
{
  return right;
};
void setRight(TreeNode *right)
{
  this->right = right;
};

int isLeaf( )
{
  if( this->left == NULL && this->right == NULL )
      return 1;
  return 0;
};
}; // end class TreeNode
```

C++ Implementation (Tree.h)

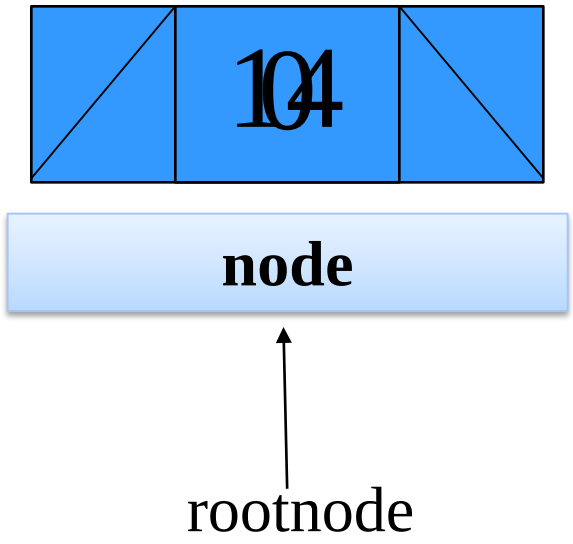
```
class Tree
{
   TreeNode * rootnode;
public:
   Tree()
   {
       rootnode=NULL;
   }
   void insert(int info);
//we will add other functions of class tree here

};
```

C++ Implementation (Static Main without Menu)

```
#include <iostream>
#include "TreeNode.h"

int main()
{
   int x[] = { 14, 15, 4, 9, 7, 18, 3, 5, 16,4, 20, 17,
               9, 14,5, -1};
   for(int i=0; x[i] > 0; i++ )
   {
        insert(x[i] );
   }
}
```

C++ Implementation(Tree.cpp)

```
Void Tree::insert(int info)
{
     TreeNode* node = new TreeNode();
     node->setinfo(info);
     if(rootnode == NULL)
     {

        rootnode=node;

     }
     else
     {
```



C++ Implementation

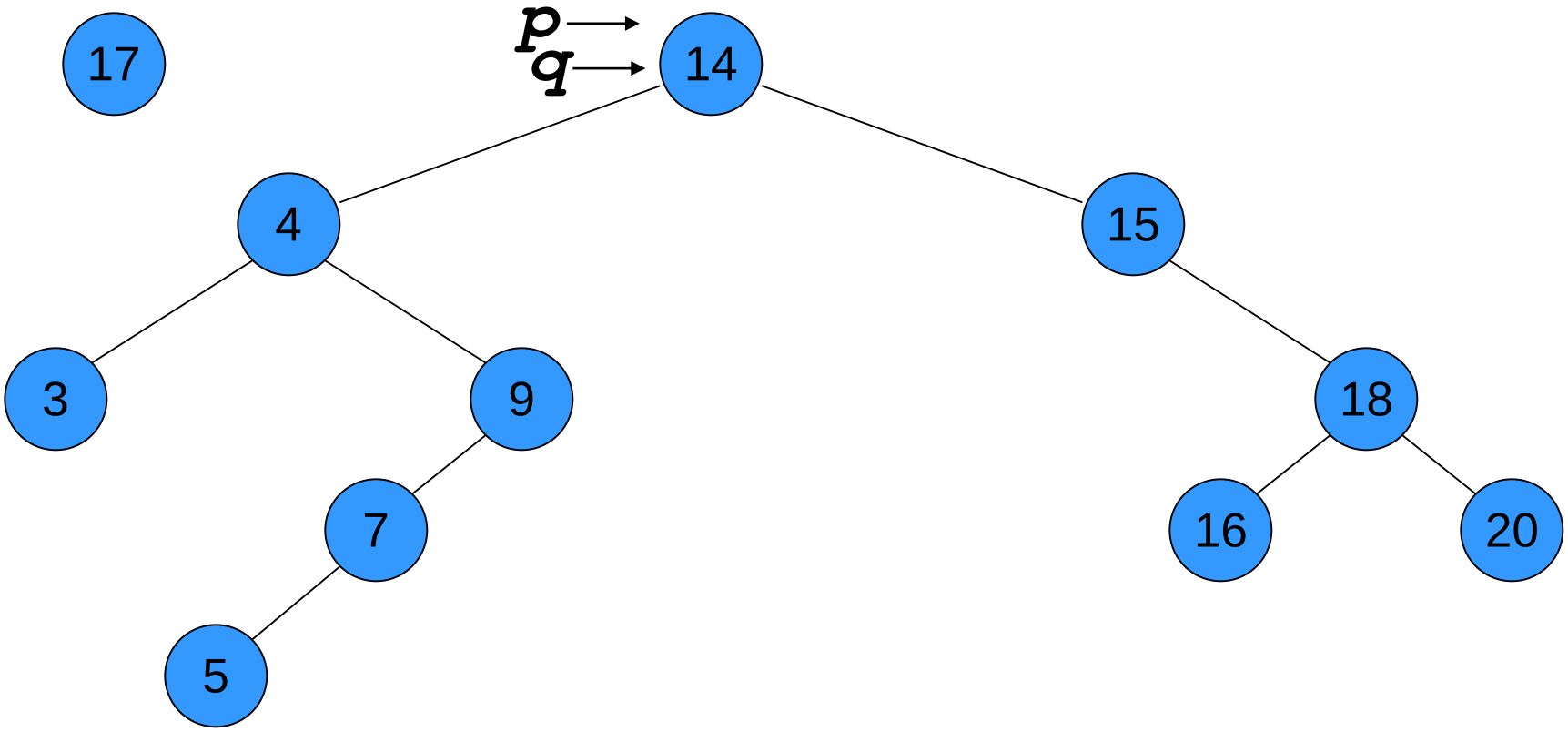
```
 //else part

TreeNode *p, *q;
p = q = rootnode;
while( info != (p->getInfo()) && q != NULL )
{
    p = q;
    if( info < (p->getInfo()) )
        q = p->getLeft();
    else
        q = p->getRight();
}
```

C++ Implementation

```
    if( info == (p->getInfo()) ){
        cout << "attempt to insert duplicate: "
             <<info << endl;
        delete node;
    }
    else if( info < (p->getInfo()) )
        p->setLeft( node );
    else
        p->setRight( node );
 }//end of else
} // end of insert
```

Trace of insert

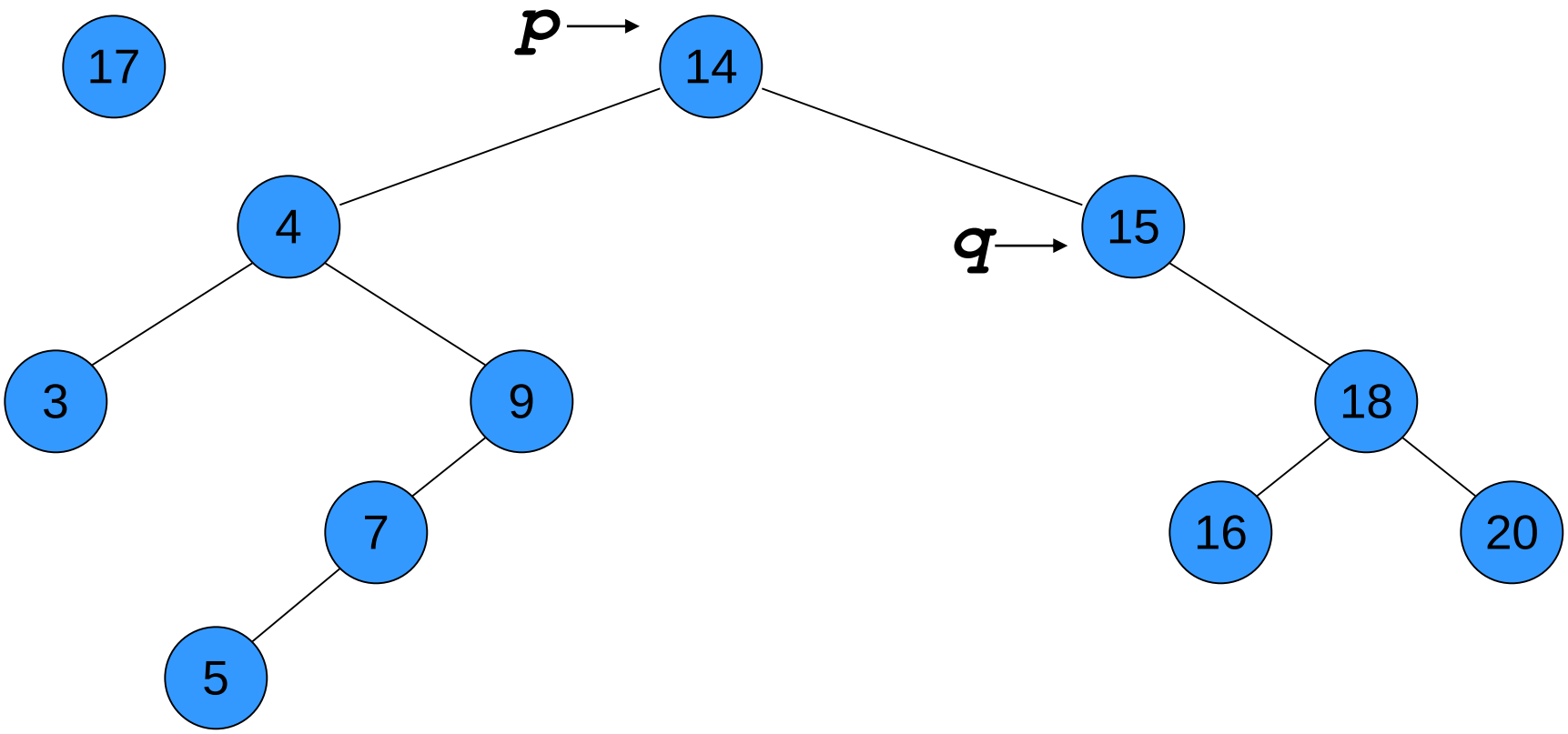
```
while( info != (p->getInfo()) && q != NULL )
    {
        p = q;
        if( info < (p->getInfo()) )
            q = p->getLeft();
        else
            q = p->getRight();
    }
```



17, 9, 14, 5

Trace of insert

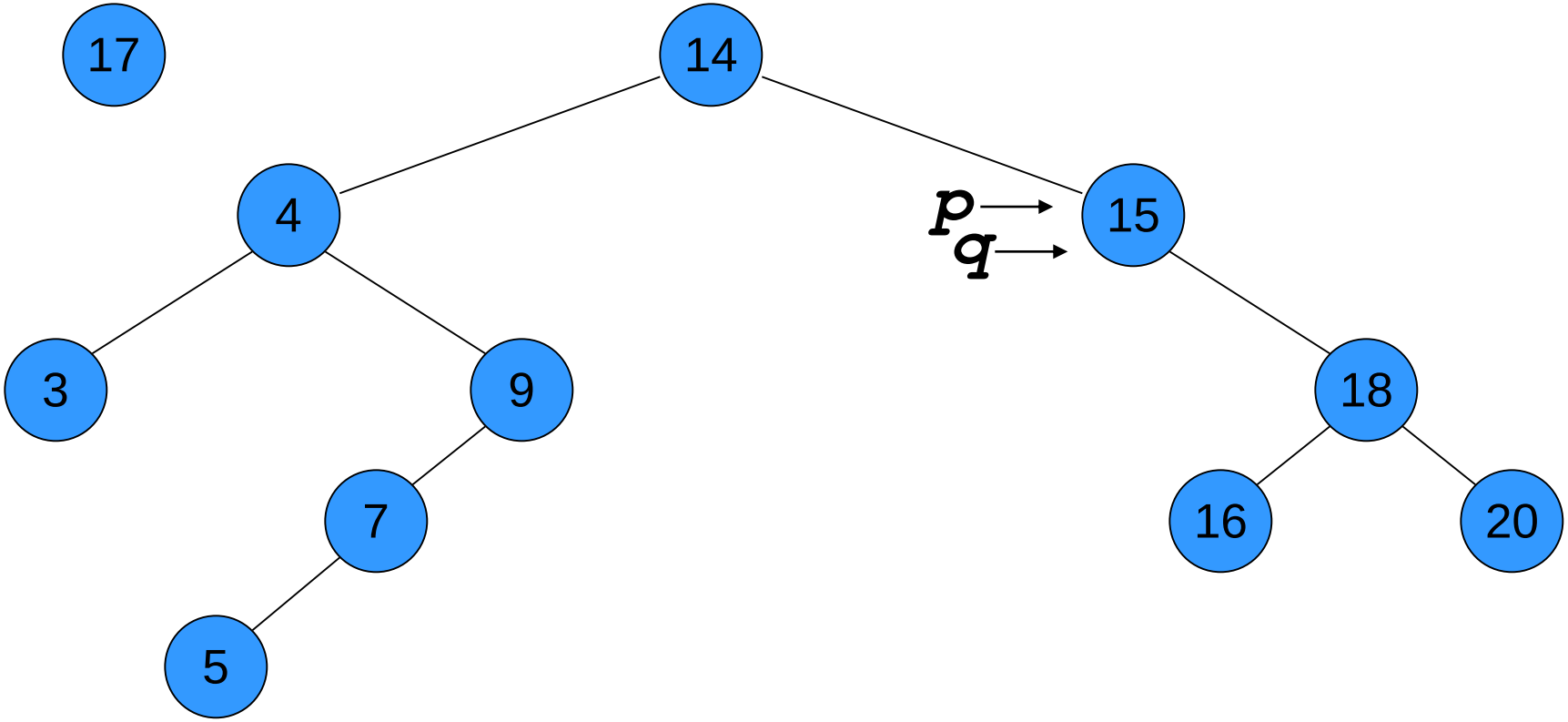
```
while( info != (p->getInfo()) && q != NULL )
    {
        p = q;
        if( info < (p->getInfo()) )
            q = p->getLeft();
        else
            q = p->getRight();
    }
```



17, 9, 14, 5

Trace of insert

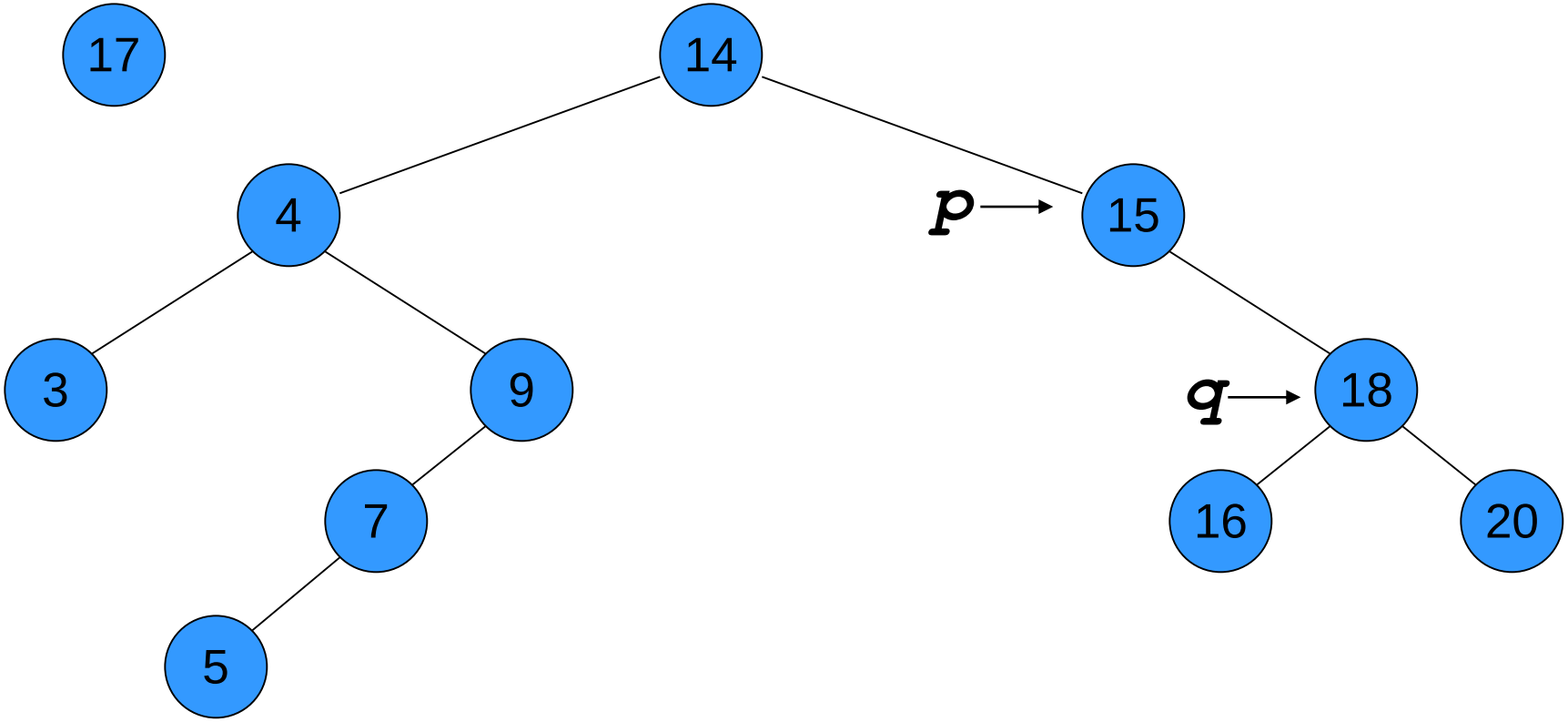
```
while( info != (p->getInfo()) && q != NULL )
    {
        p = q;
        if( info < (p->getInfo()) )
            q = p->getLeft();
        else
            q = p->getRight();
    }
```



17, 9, 14, 5

Trace of insert

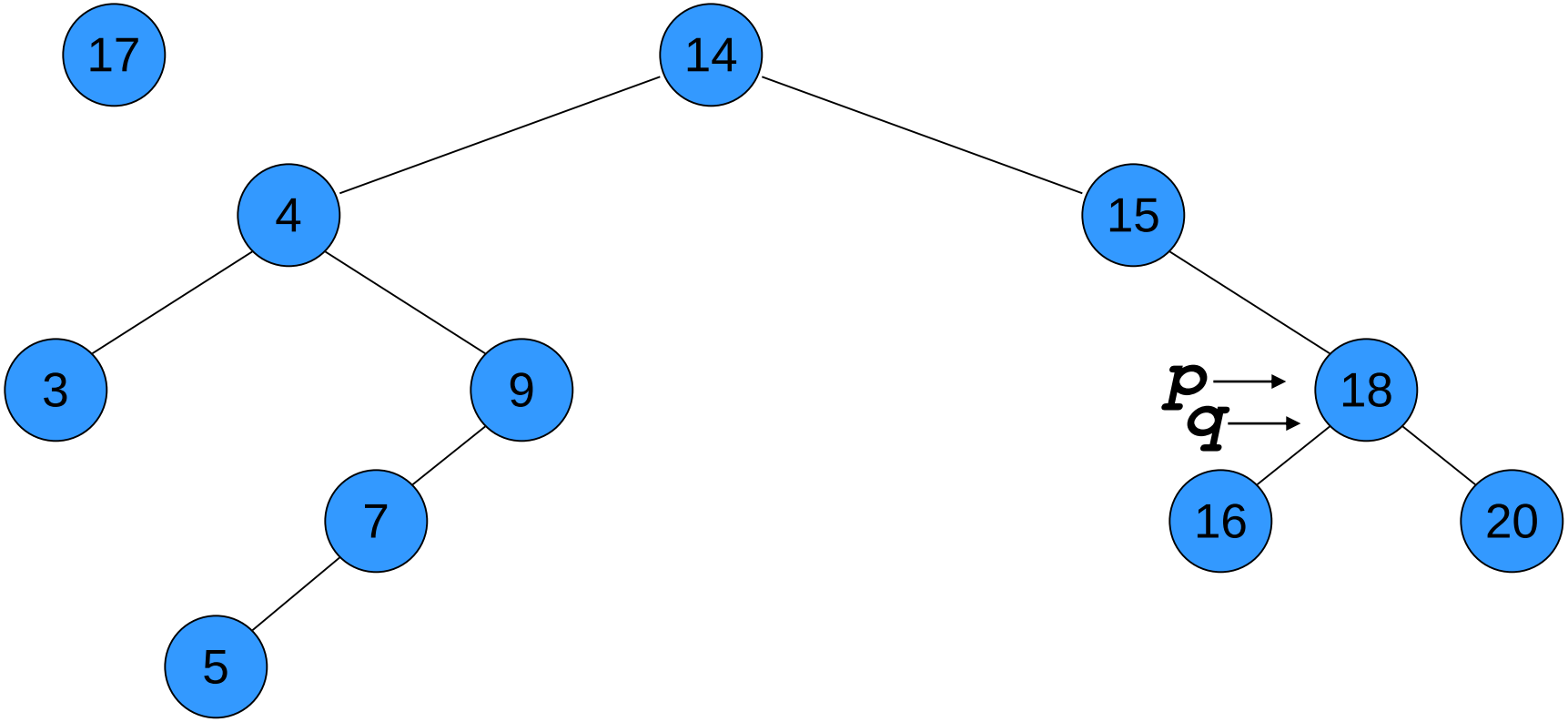
```
while( info != (p->getInfo()) && q != NULL )
    {
        p = q;
        if( info < (p->getInfo()) )
            q = p->getLeft();
        else
            q = p->getRight();
    }
```



17, 9, 14, 5

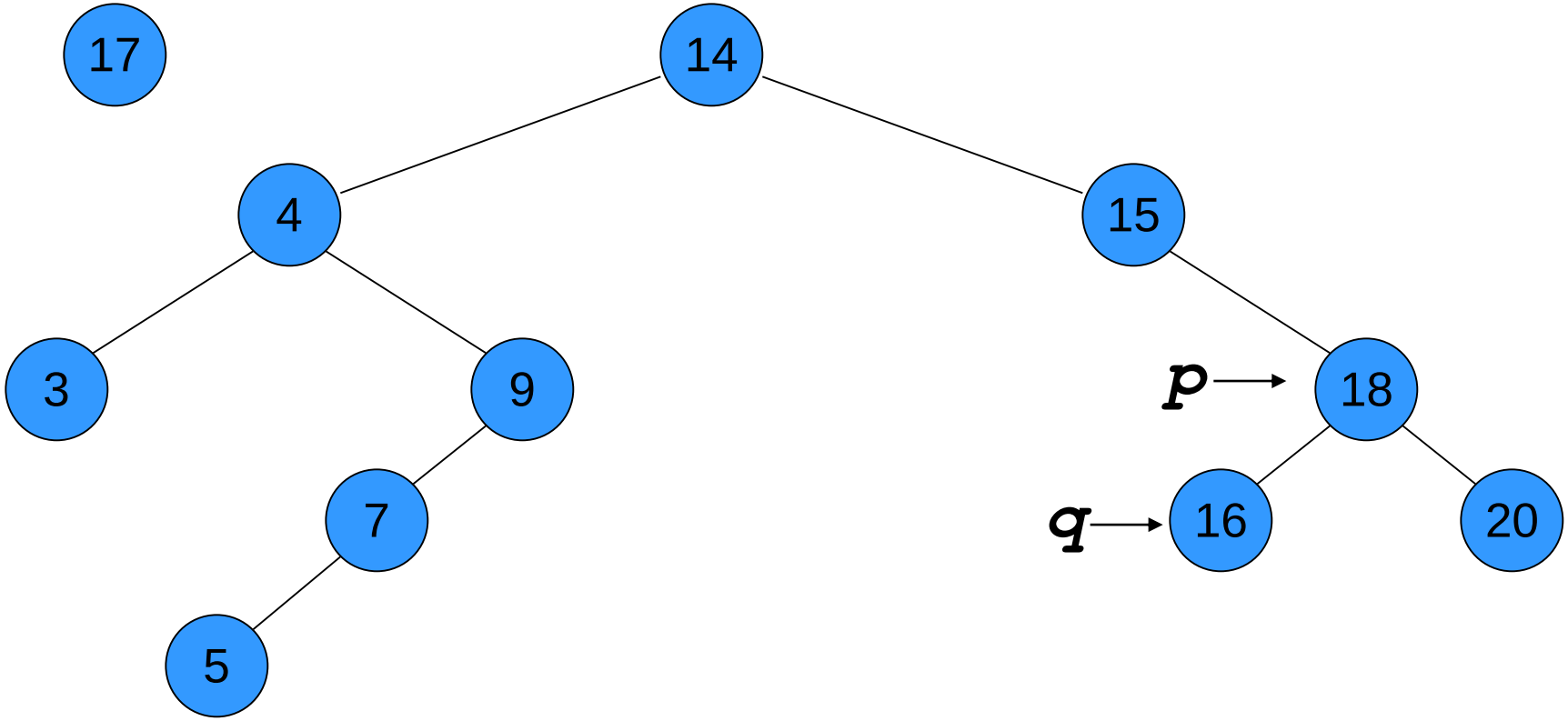
Trace of insert

```
while( info != (p->getInfo()) && q != NULL )
    {
        p = q;
        if( info < (p->getInfo()) )
            q = p->getLeft();
        else
            q = p->getRight();
    }
```



17, 9, 14, 5

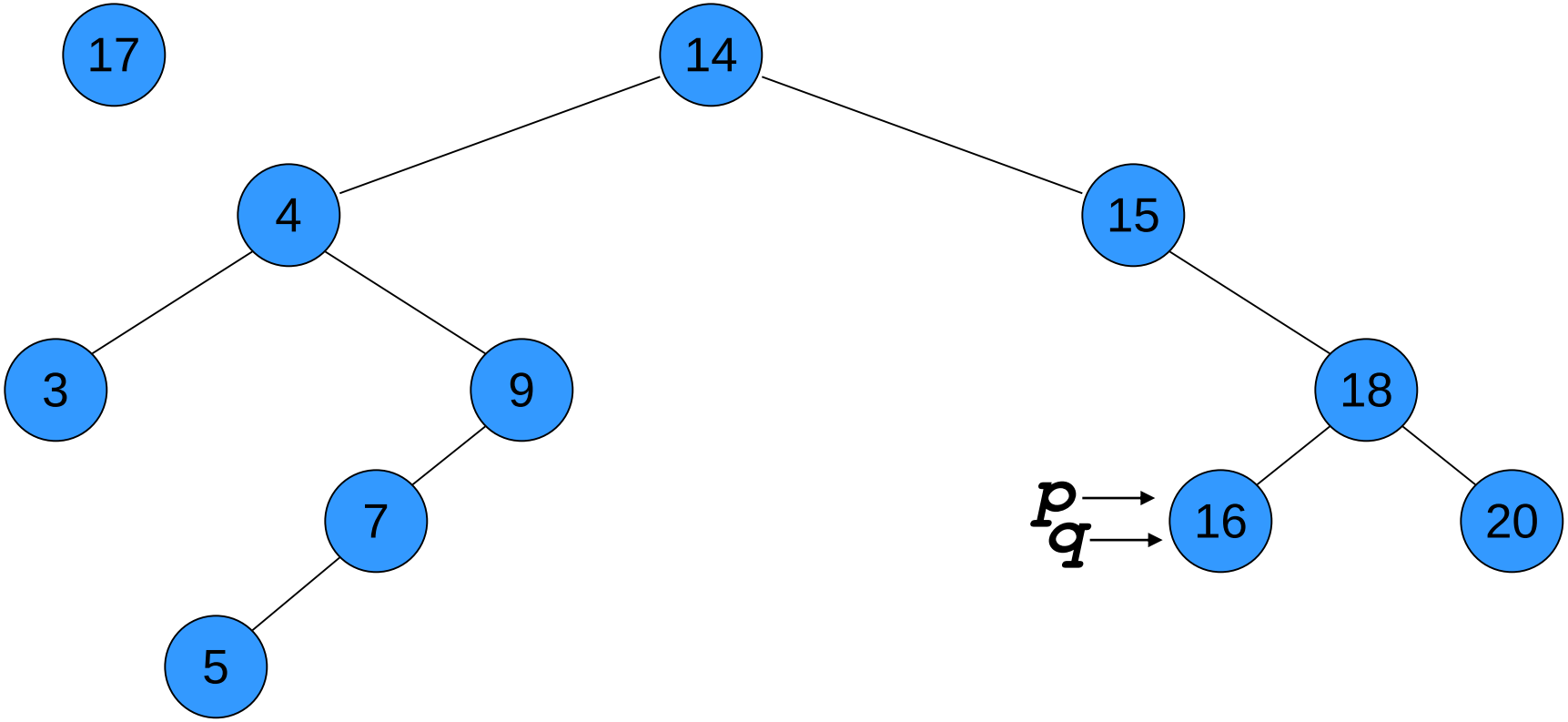
Trace of insert



17, 9, 14, 5

Trace of insert

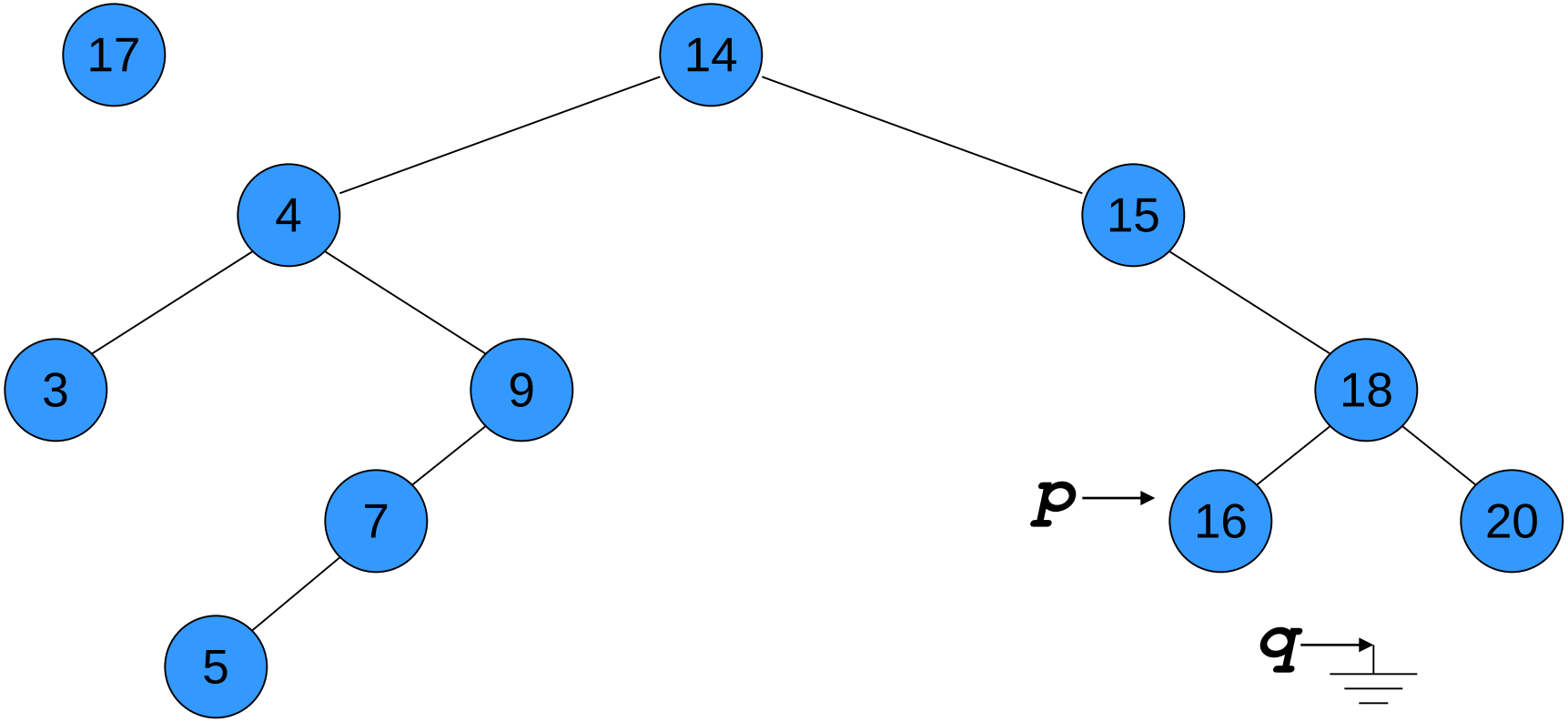
```
while( info != (p->getInfo()) && q != NULL )
    {
        p = q;
        if( info < (p->getInfo()) )
            q = p->getLeft();
        else
            q = p->getRight();
    }
```



17, 9, 14, 5

Trace of insert

```
while( info != (p->getInfo()) && q != NULL )
    {
        p = q;
        if( info < (p->getInfo()) )
            q = p->getLeft();
        else
            q = p->getRight();
    }
```

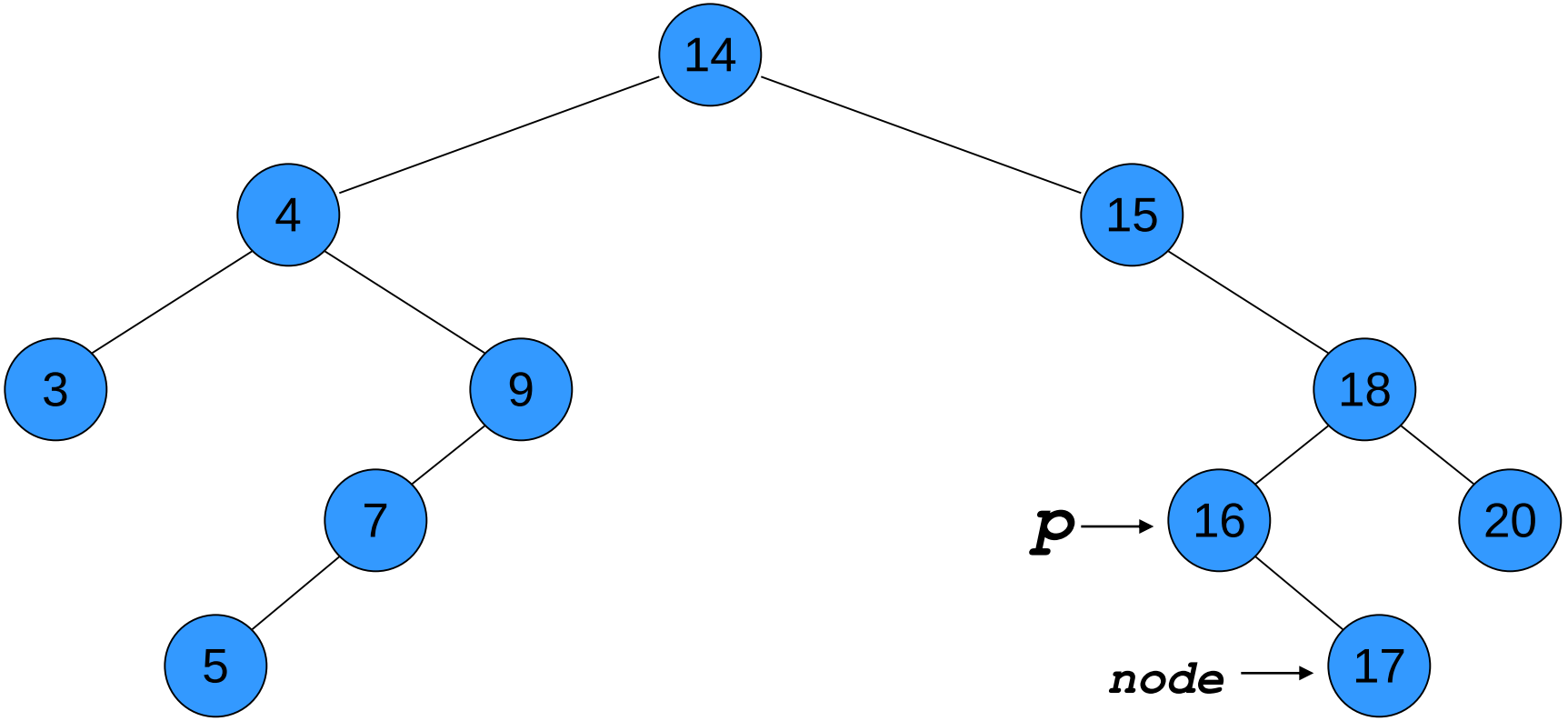


17, 9, 14, 5

Trace of insert

```
while( info != (p->getInfo()) && q != NULL )
{
    p = q;
    if( info < (p->getInfo()) )
        q = p->getLeft();
    else
        q = p->getRight();
}
```

```
else if( info < (p->getInfo()) )
    p->setLeft( node );
else
    p->setRight( node );
```



17, 9, 14, 5

```
p->setRight( node );
```

Cost of Search

- Given that a binary tree is level *d* deep. How long does it take to find out whether a number is already present?
- Consider the insert(17) in the example tree.
- Each time around the while loop, we did one comparison.
- After the comparison, we moved a level *down*.

Cost of Search

- With the binary tree in place, we can write a routine *find(x)* that returns true if the number *x* is present in the tree, false otherwise.
- How many comparison are needed to find out if *x* is present in the tree?
- We do one comparison at each level of the tree until either *x* is found or q becomes NULL.

Cost of Search

- If the binary tree is built out of *n* numbers, how many comparisons are needed to find out if a number *x* is in the tree?
- Recall that the depth of the complete binary tree built using '*n*' nodes will be $log_2(n+1) - 1$.
- For example, for n=100,000, $\log_2(100001)$ is less than 20; the tree would be 20 levels deep.

Cost of Search

- If the tree is complete binary or nearly complete, searching through 100,000 numbers will require a maximum of 20 comparisons.
- Or in general, approximately $log_2(n)$.
- Compare this with a linked list of 100,000 numbers. The comparisons required could be a maximum of *n*.