

ECE 220

Lecture x0018 - 11/21
BSTs and C++ examples

Binary Search Trees

Binary Search Trees

- Binary trees that have a particular *sorted* property are called **binary search trees** (BST)

Binary Search Trees

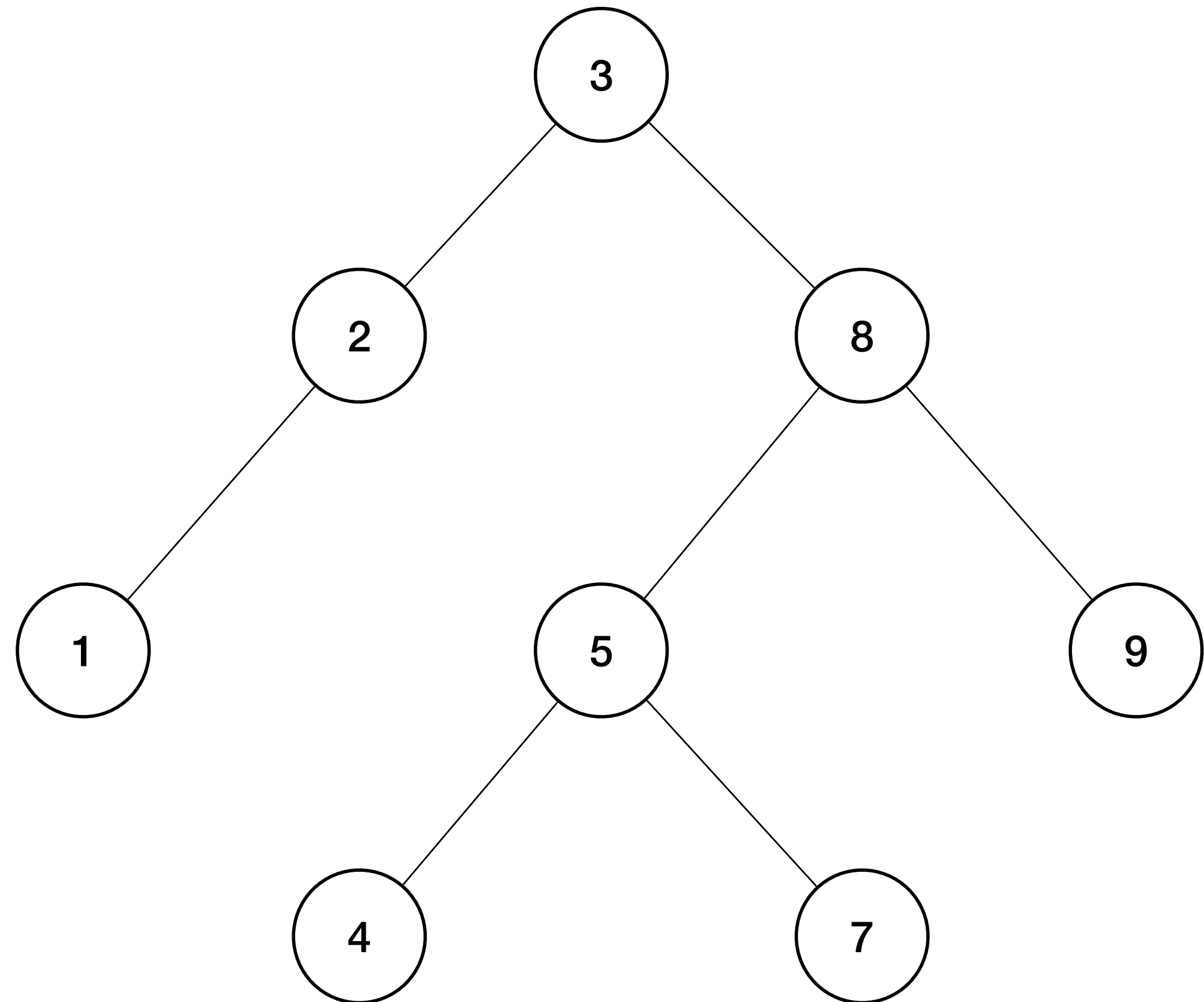
- Binary trees that have a particular *sorted* property are called **binary search trees (BST)**
 - All nodes in the **left subtree** of a given node are *lesser than or equal* to the node

Binary Search Trees

- Binary trees that have a particular *sorted* property are called **binary search trees (BST)**
 - All nodes in the **left subtree** of a given node are lesser than or equal to the node
 - All nodes in the **right subtree** of a given node are greater than that node

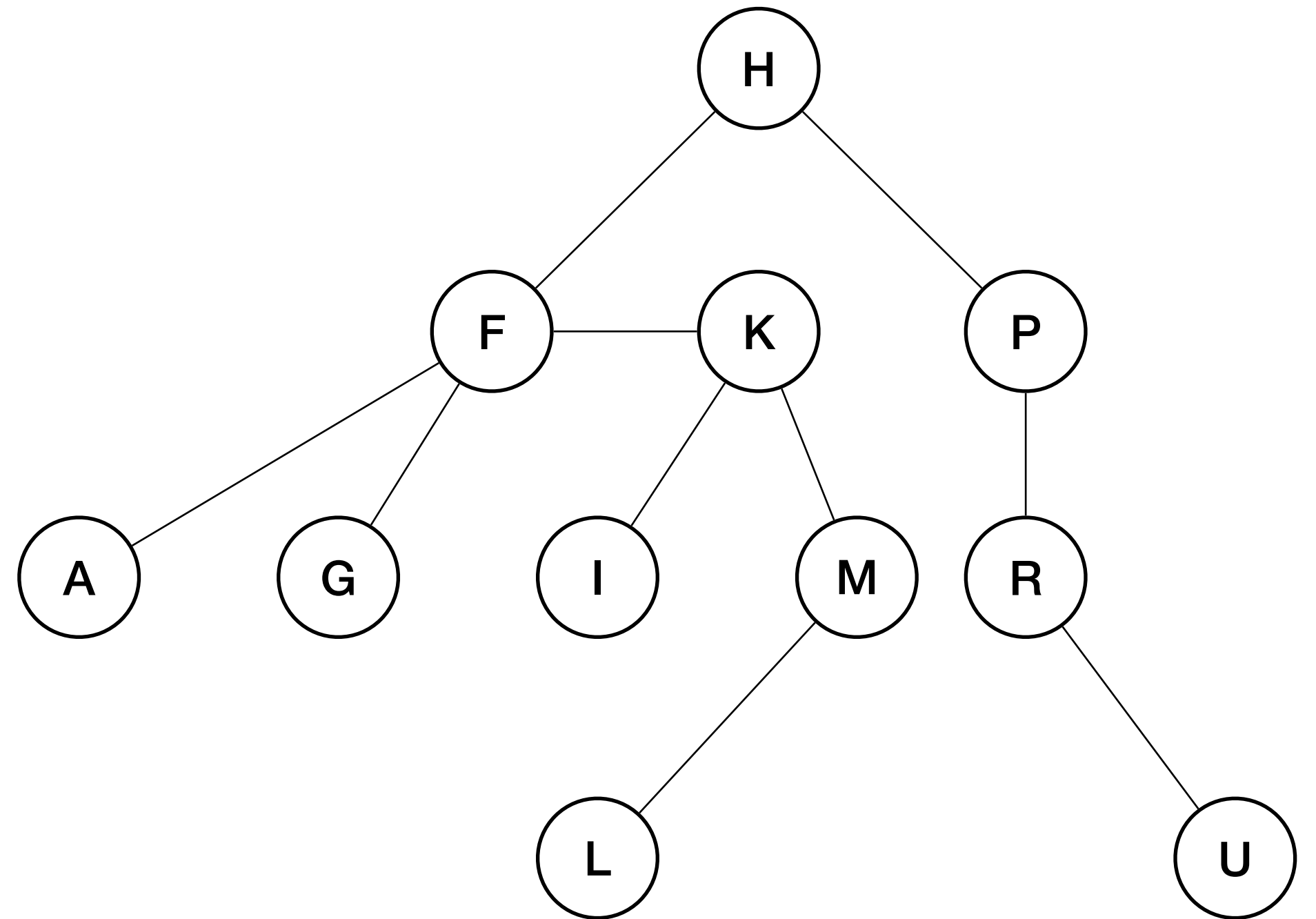
Binary Search Trees

- Binary trees that have a particular *sorted* property are called **binary search trees (BST)**
 - All nodes in the **left subtree** of a given node are lesser than or equal to the node
 - All nodes in the **right subtree** of a given node are greater than that node



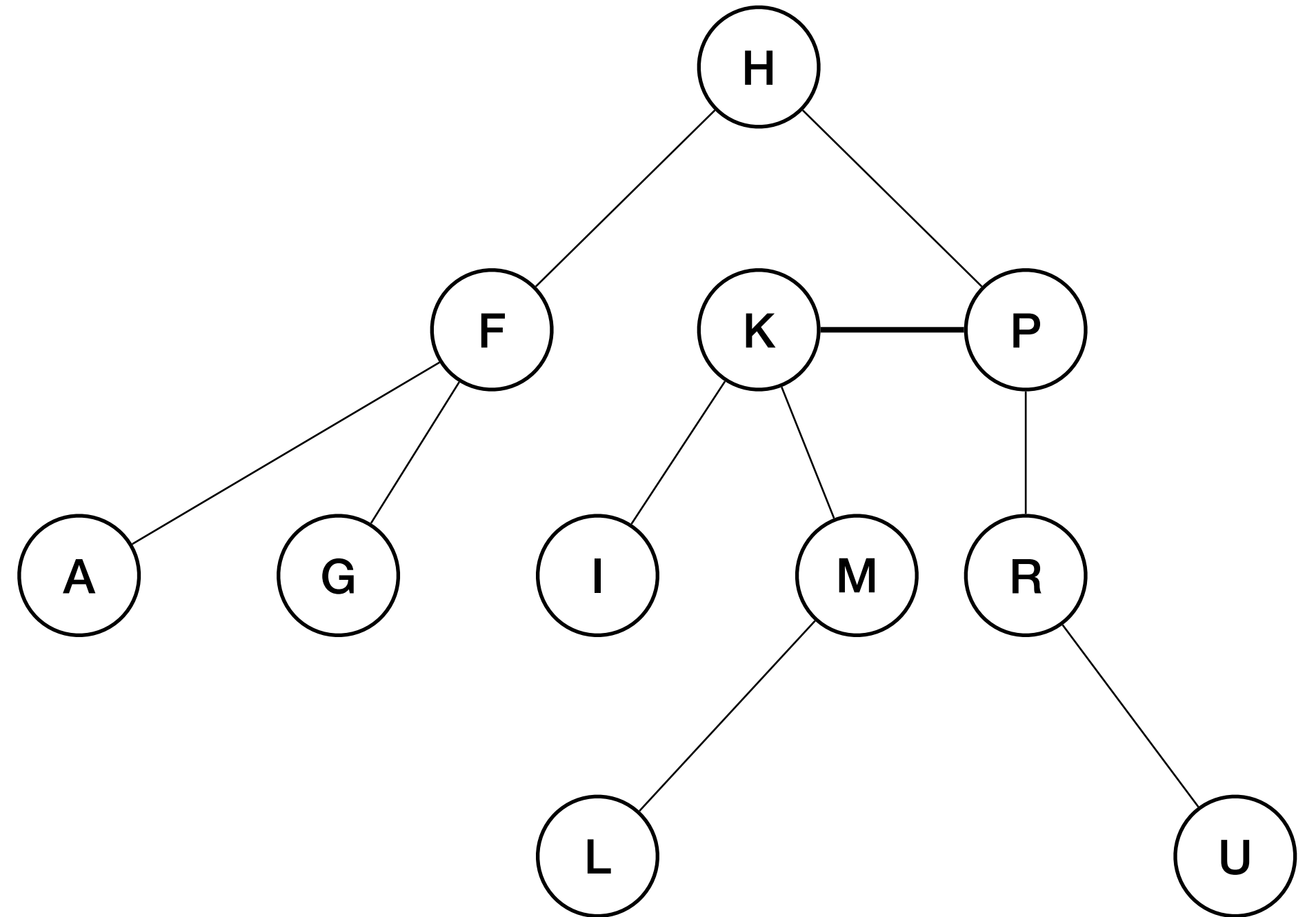
Concept check

- Who are the siblings of R?
- What is the depth of node I?
- List the leaf nodes?
- What is the height of the tree?
- Is this a Binary Search Tree?



Concept check

- Who are the siblings of R?
- What is the depth of node I?
- List the leaf nodes?
- What is the height of the tree?
- Is this a Binary Search Tree?



Exercises with BST

Exercises with BST

- How can you find the minimum or maximum element in a BST?

Exercises with BST

- How can you find the minimum or maximum element in a BST?
- How can we search a BST for a node?

Exercises with BST

- How can you find the minimum or maximum element in a BST?
- How can we search a BST for a node?
- How should you insert a new node in a BST?

Exercises with BST

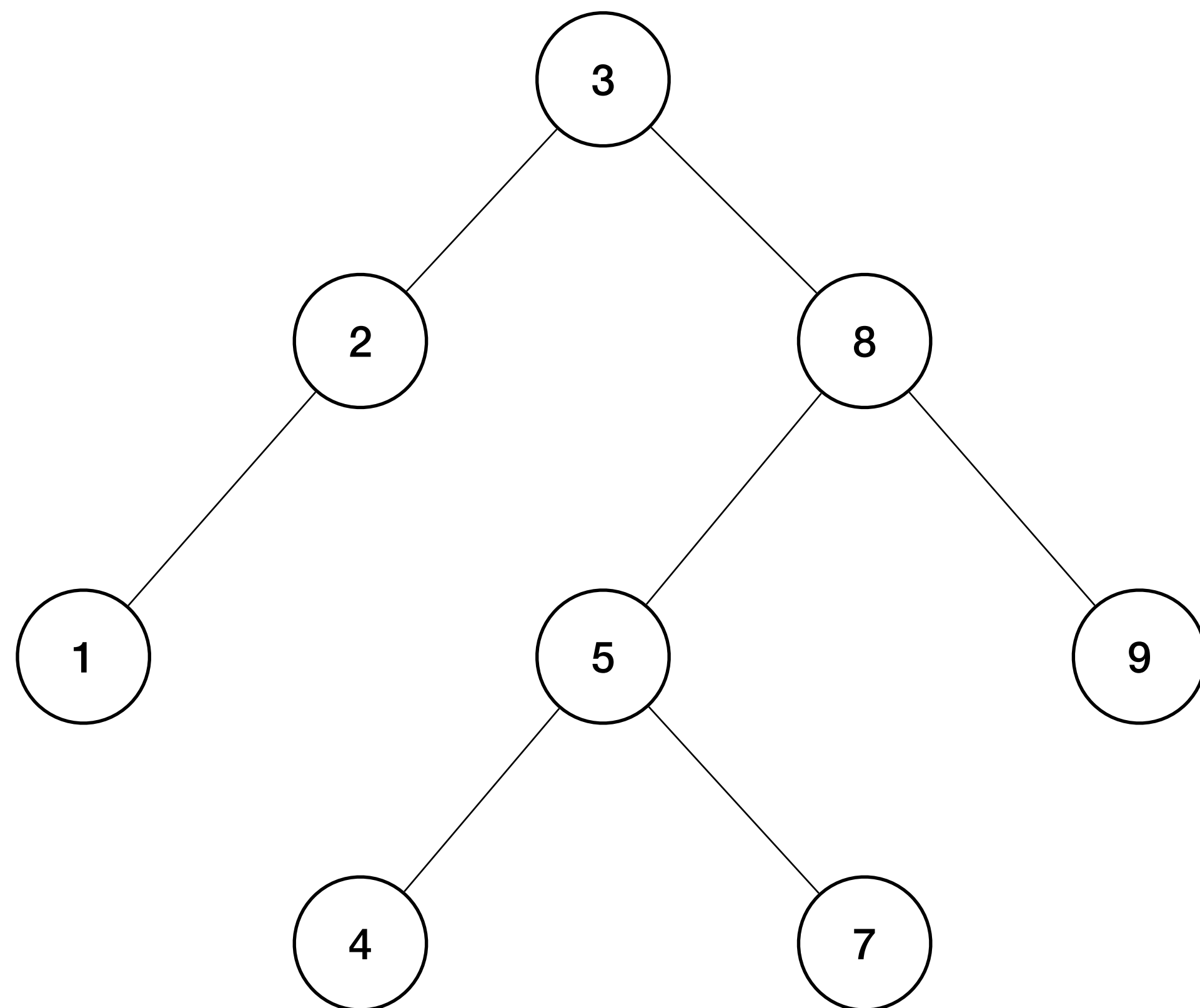
- How can you find the minimum or maximum element in a BST?
- How can we search a BST for a node?
- How should you insert a new node in a BST?
- How can you find the height of a *general* tree (can also be BST)?

```
typedef struct node{
    int data;
    struct node *left;
    struct node *right;
} node;
```

Finding extremals in a BST

Minimum - keep going left

Minimum - keep going right



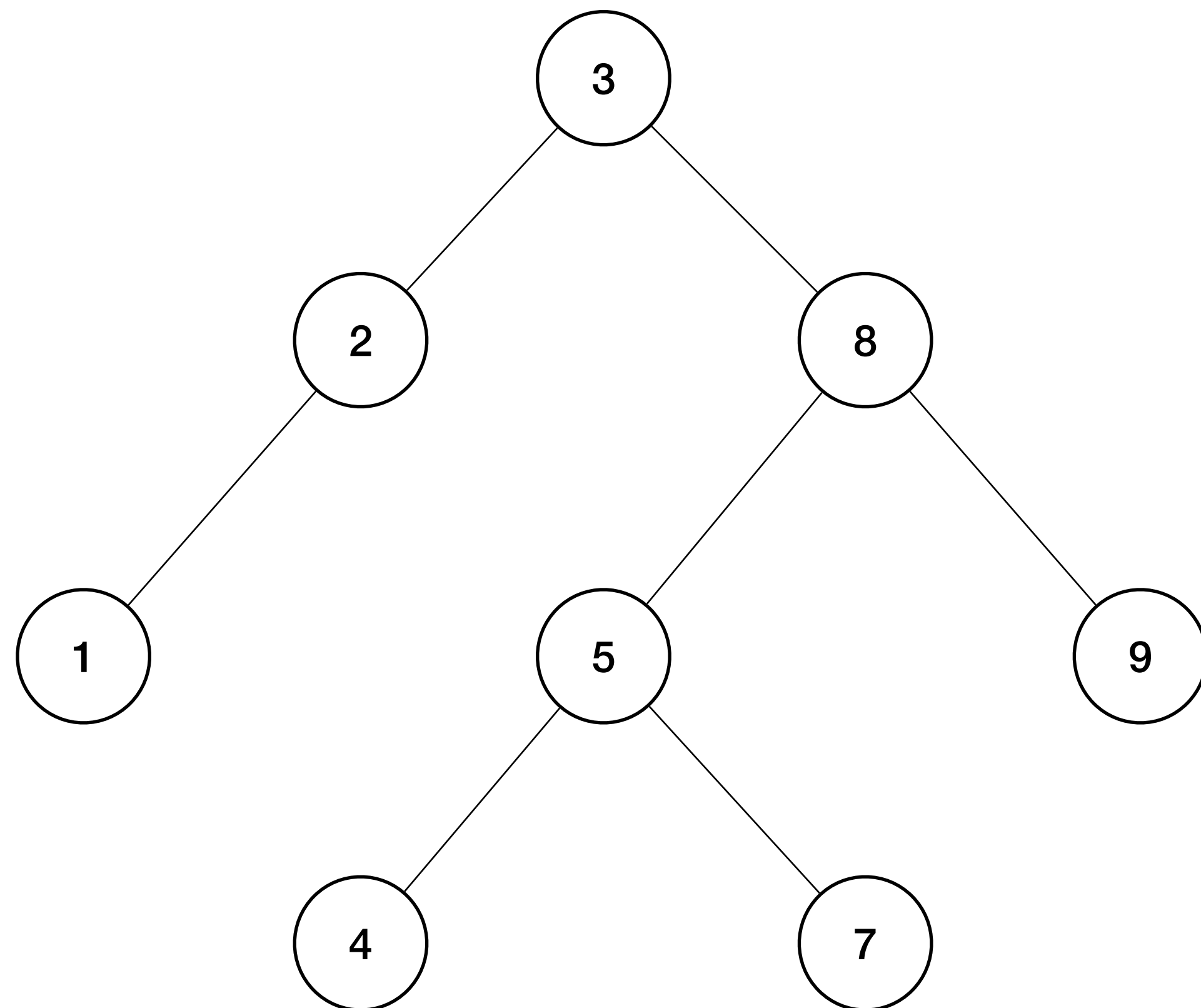

```
typedef struct node{
    int data;
    struct node *left;
    struct node *right;
} node;
```

Finding extremals in a BST

Minimum - keep going left

```
node * findmin(node *cursor){
    if (cursor->left==NULL)
        return cursor;
    else
        return findmin(cursor->left);
}
```

Minimum - keep going right



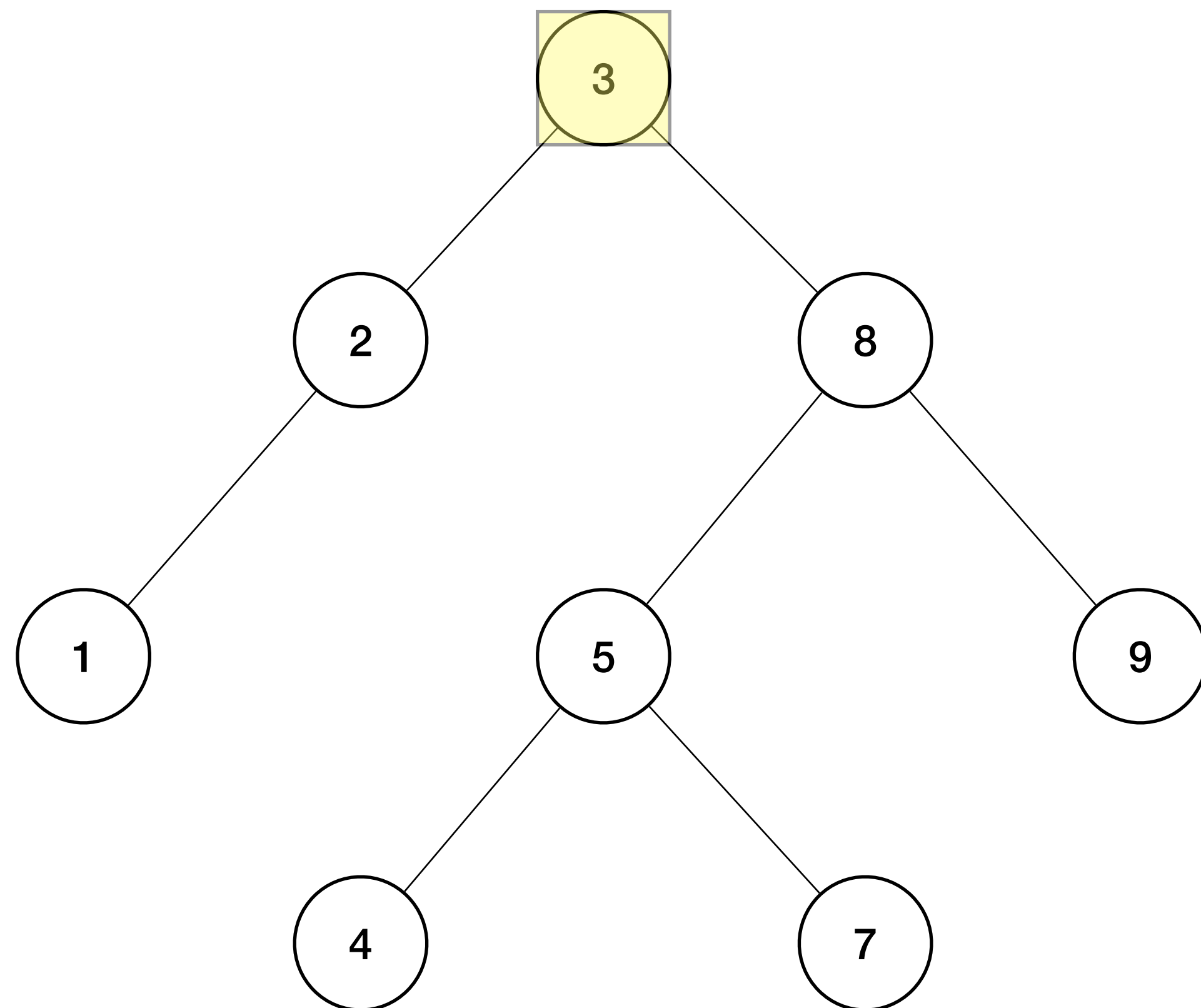
```
typedef struct node{
    int data;
    struct node *left;
    struct node *right;
} node;
```

Finding extremals in a BST

Minimum - keep going left

```
node * findmin(node *cursor){
    if (cursor->left==NULL)
        return cursor;
    else
        return findmin(cursor->left);
}
```

Minimum - keep going right



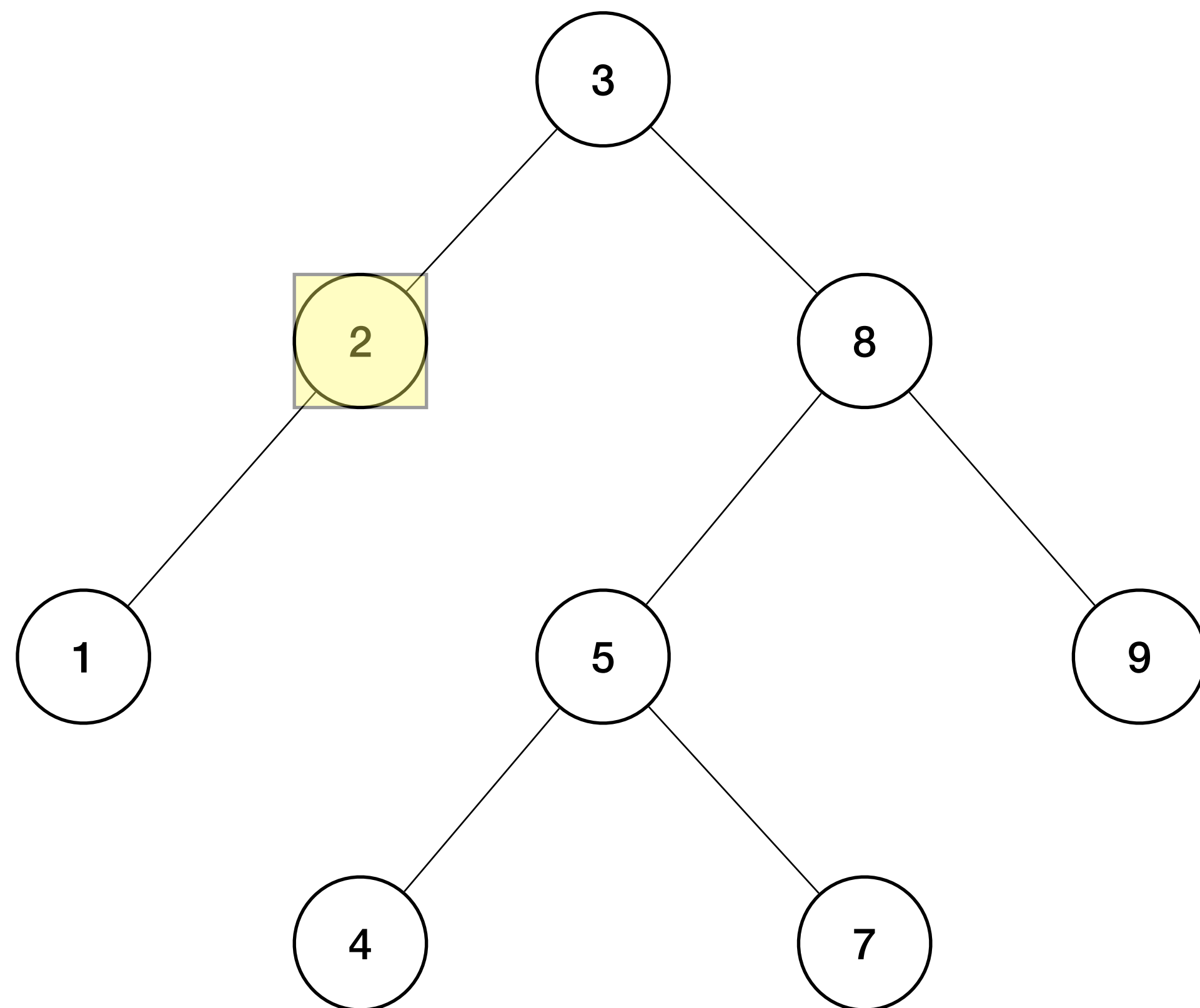
```
typedef struct node{
    int data;
    struct node *left;
    struct node *right;
} node;
```

Finding extremals in a BST

Minimum - keep going left

```
node * findmin(node *cursor){
    if (cursor->left==NULL)
        return cursor;
    else
        return findmin(cursor->left);
}
```

Minimum - keep going right



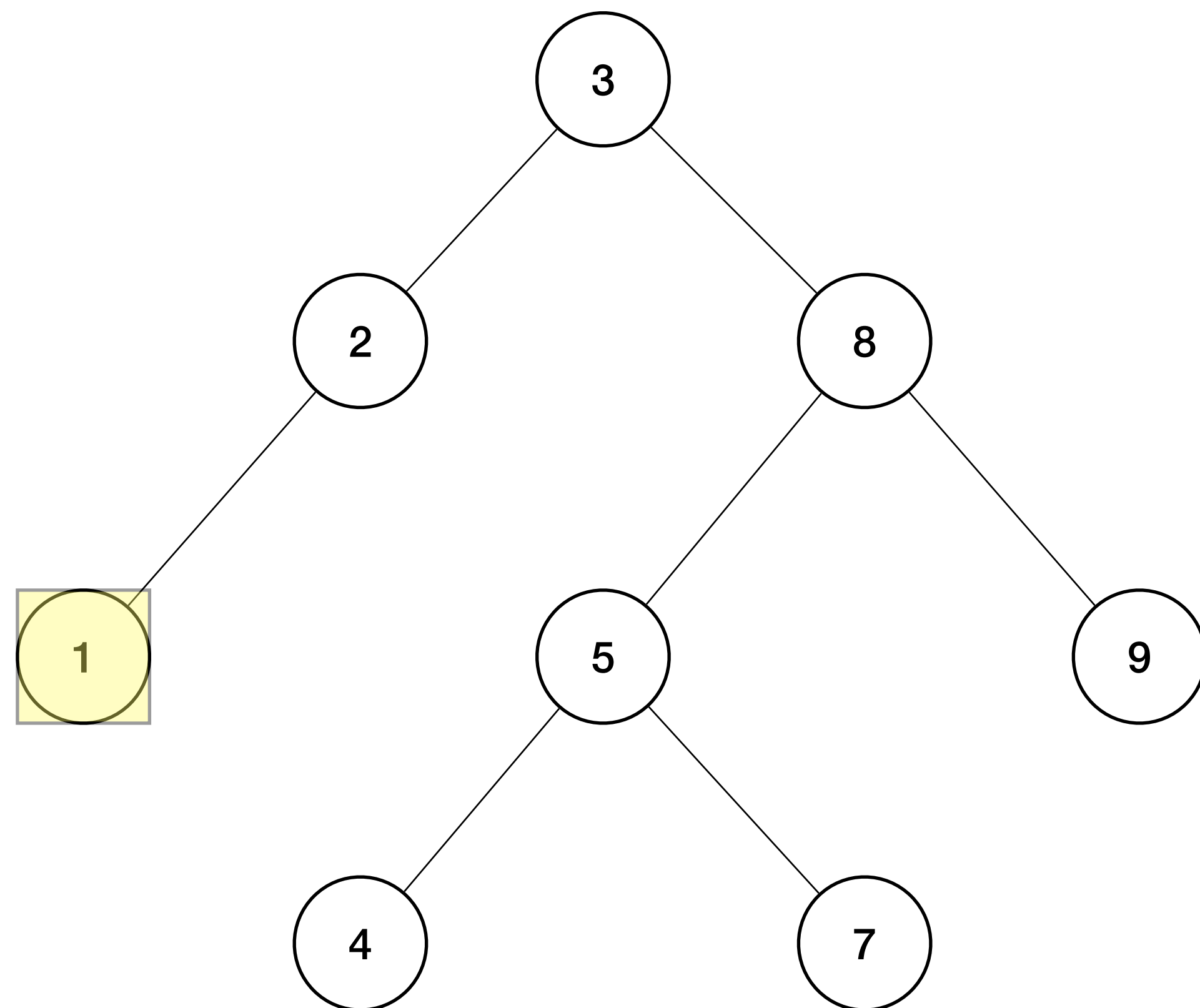

```
typedef struct node{
    int data;
    struct node *left;
    struct node *right;
} node;
```

Finding extremals in a BST

Minimum - keep going left

```
node * findmin(node *cursor){
    if (cursor->left==NULL)
        return cursor;
    else
        return findmin(cursor->left);
}
```

Minimum - keep going right



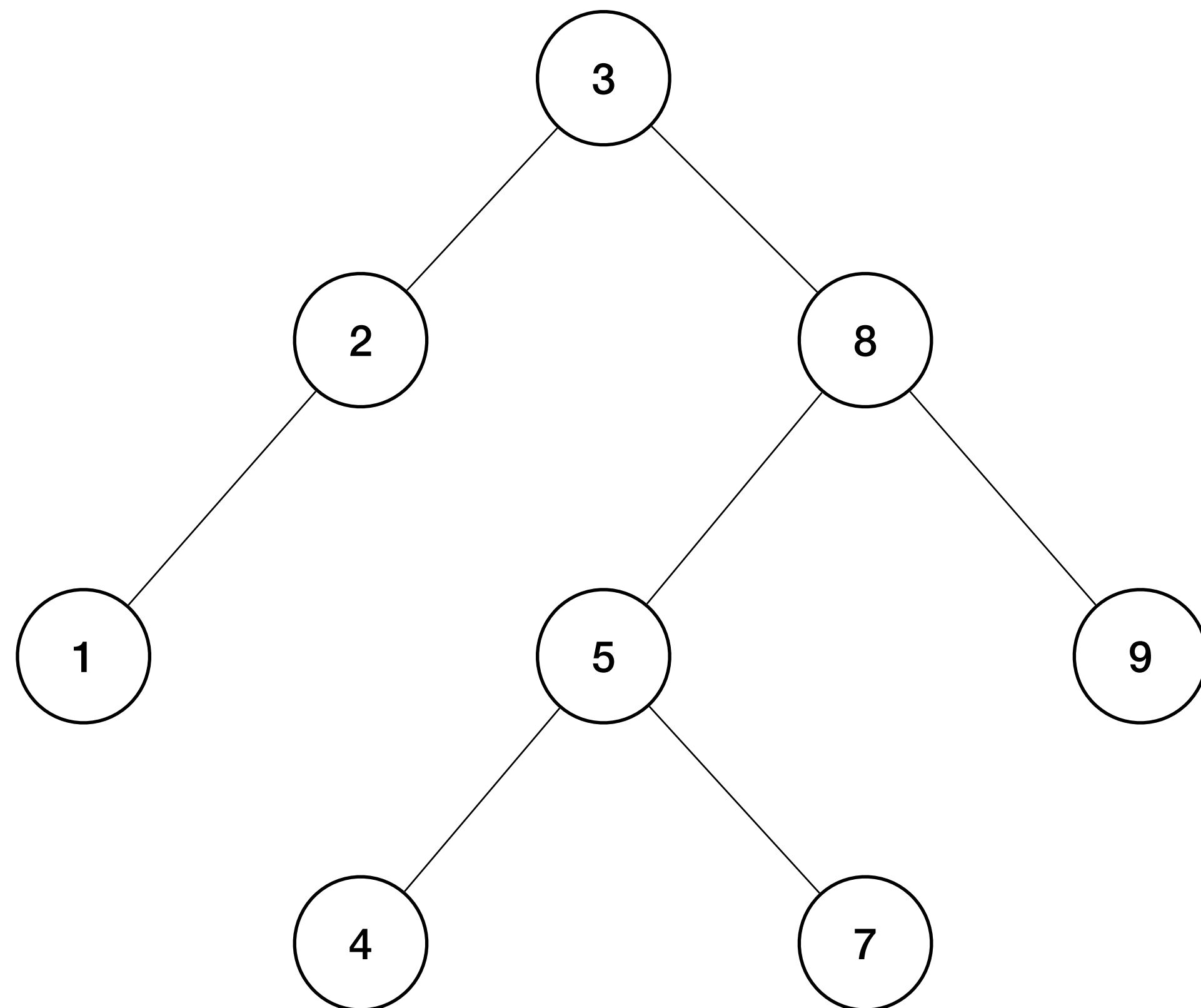
```
typedef struct node{
    int data;
    struct node *left;
    struct node *right;
} node;
```

Finding extremals in a BST

Minimum - keep going left

```
node * findmin(node *cursor){
    if (cursor->left==NULL)
        return cursor;
    else
        return findmin(cursor->left);
}
```

Minimum - keep going right



```
typedef struct node{
    int data;
    struct node *left;
    struct node *right;
} node;
```

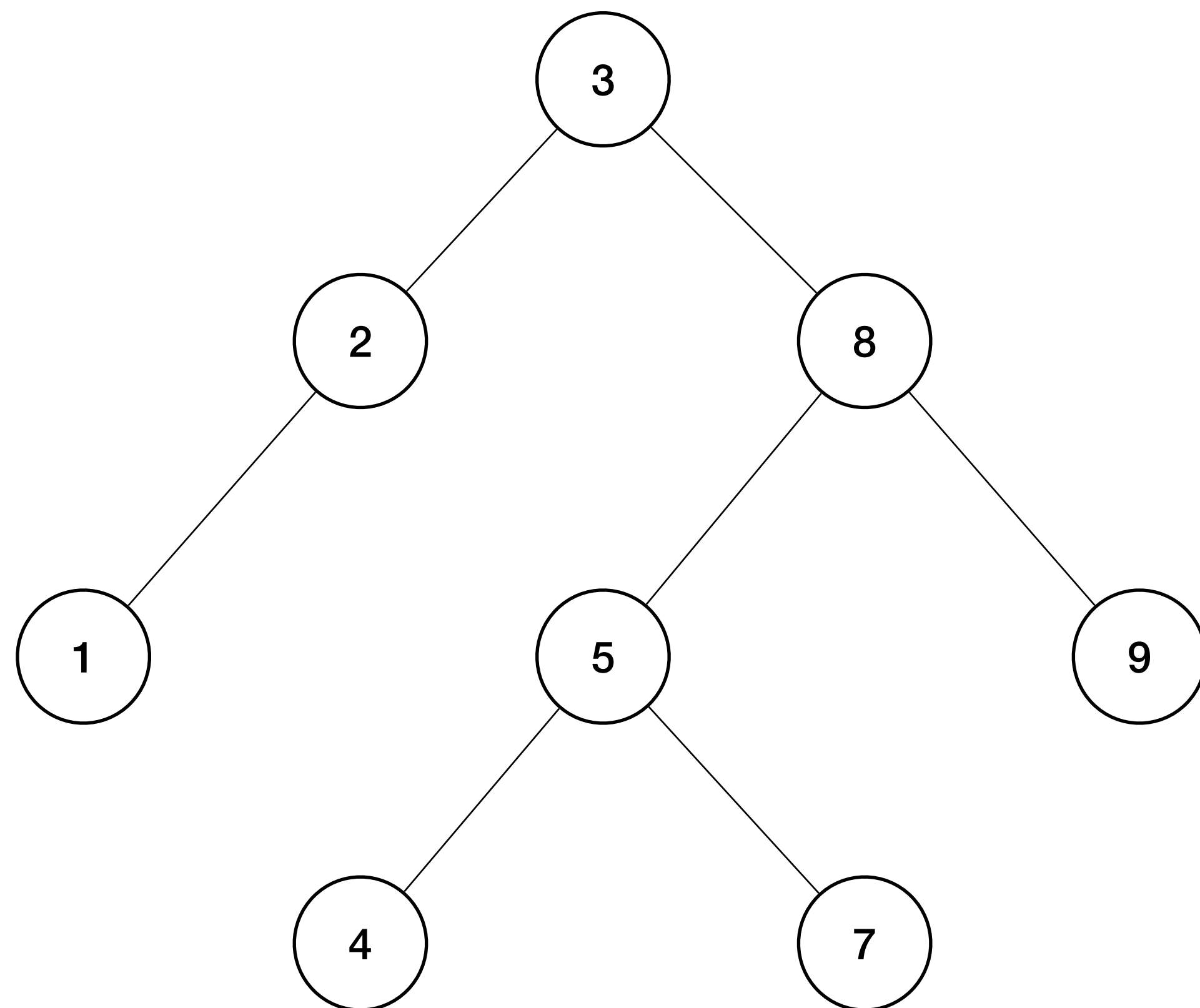
Finding extremals in a BST

Minimum - keep going left

```
node * findmin(node *cursor){
    if (cursor->left==NULL)
        return cursor;
    else
        return findmin(cursor->left);
}
```

Maximum - keep going right

```
node * findmax(node *cursor){
    if (cursor->right==NULL)
        return cursor;
    else
        return findmax(cursor->right);
}
```




```
typedef struct node{
    int data;
    struct node *left;
    struct node *right;
} node;
```

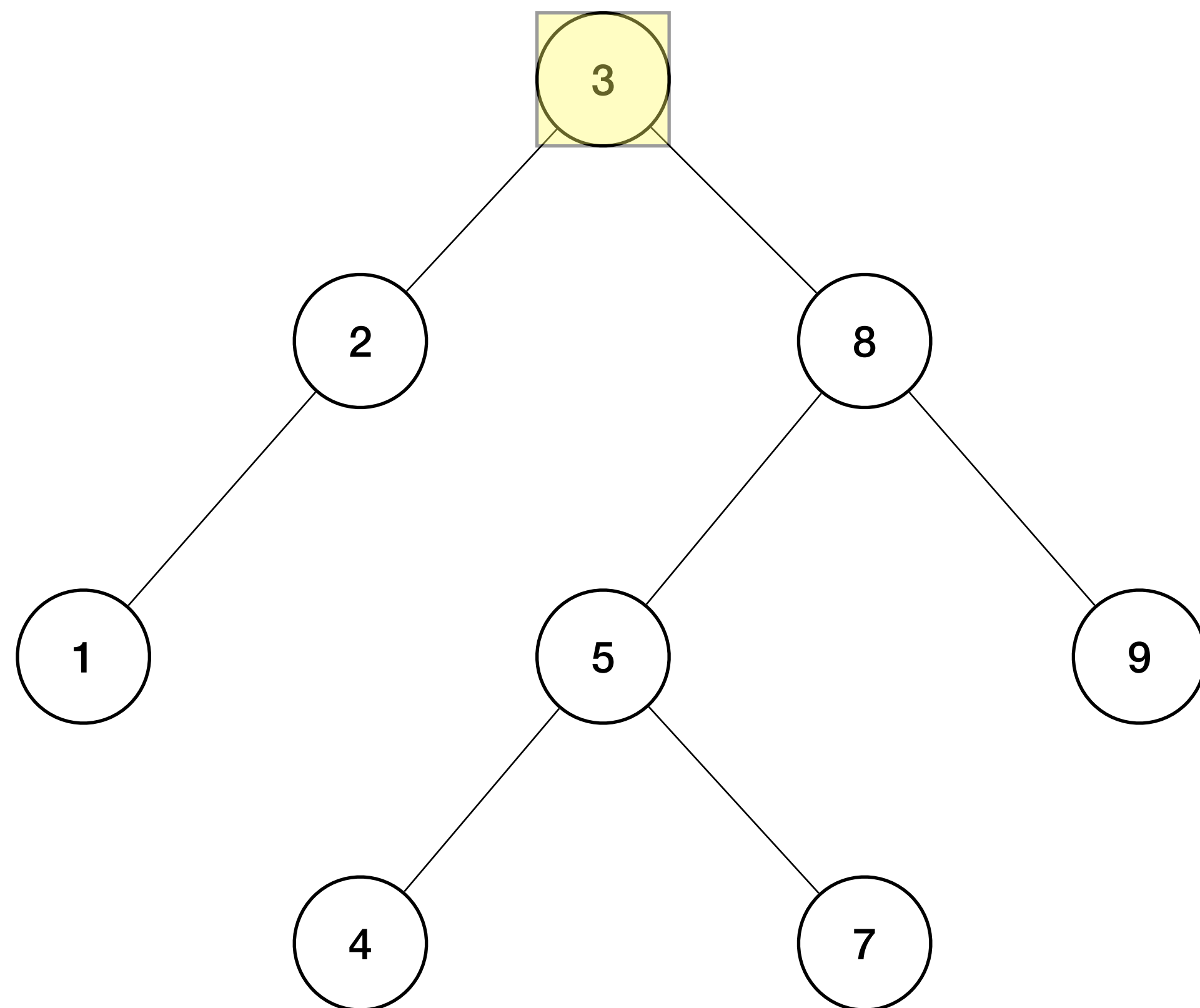
Finding extremals in a BST

Minimum - keep going left

```
node * findmin(node *cursor){
    if (cursor->left==NULL)
        return cursor;
    else
        return findmin(cursor->left);
}
```

Minimum - keep going right

```
node * findmax(node *cursor){
    if (cursor->right==NULL)
        return cursor;
    else
        return findmax(cursor->right);
}
```



```
typedef struct node{
    int data;
    struct node *left;
    struct node *right;
} node;
```

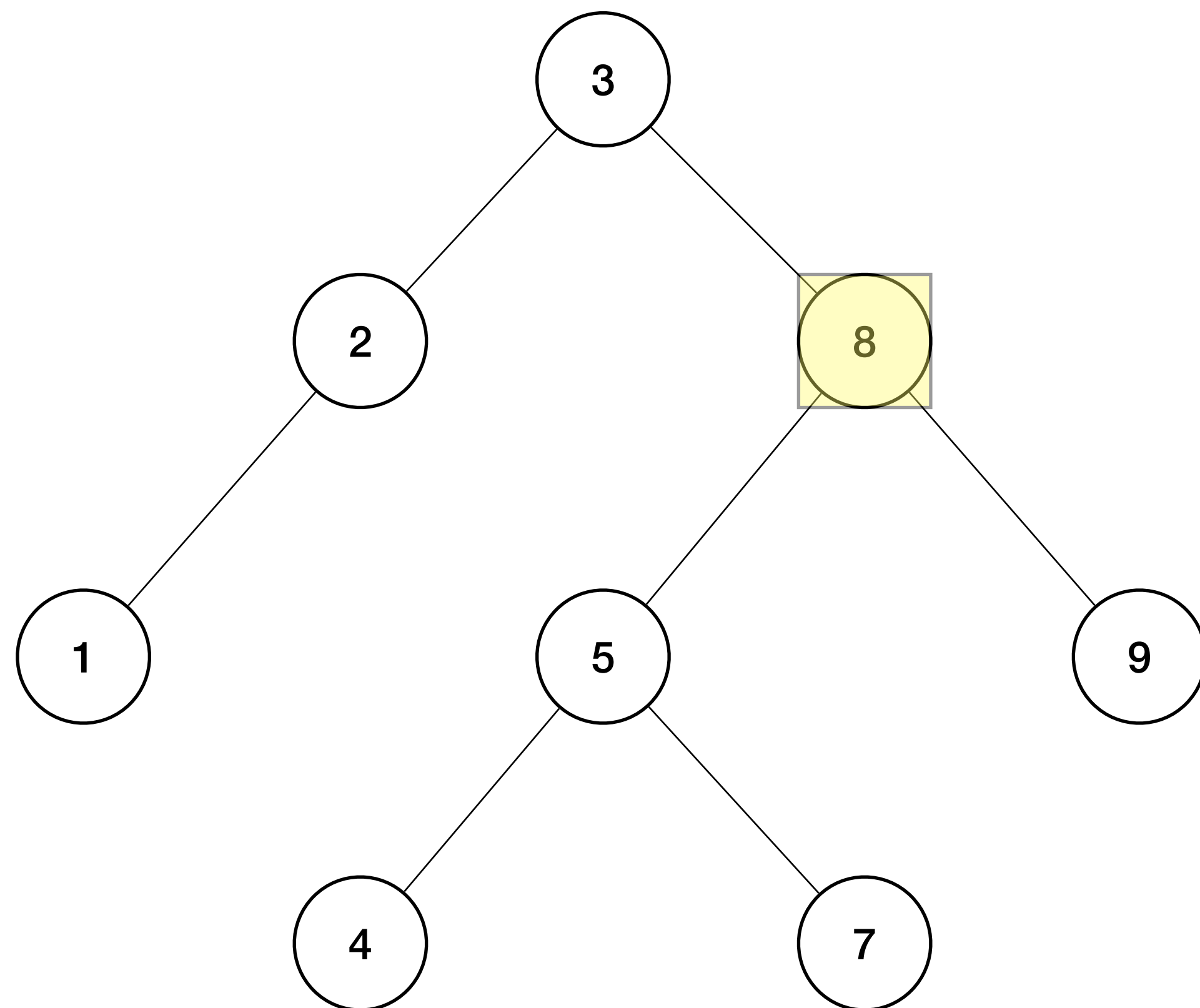
Finding extremals in a BST

Minimum - keep going left

```
node * findmin(node *cursor){
    if (cursor->left==NULL)
        return cursor;
    else
        return findmin(cursor->left);
}
```

Minimum - keep going right

```
node * findmax(node *cursor){
    if (cursor->right==NULL)
        return cursor;
    else
        return findmax(cursor->right);
}
```



```
typedef struct node{
    int data;
    struct node *left;
    struct node *right;
} node;
```

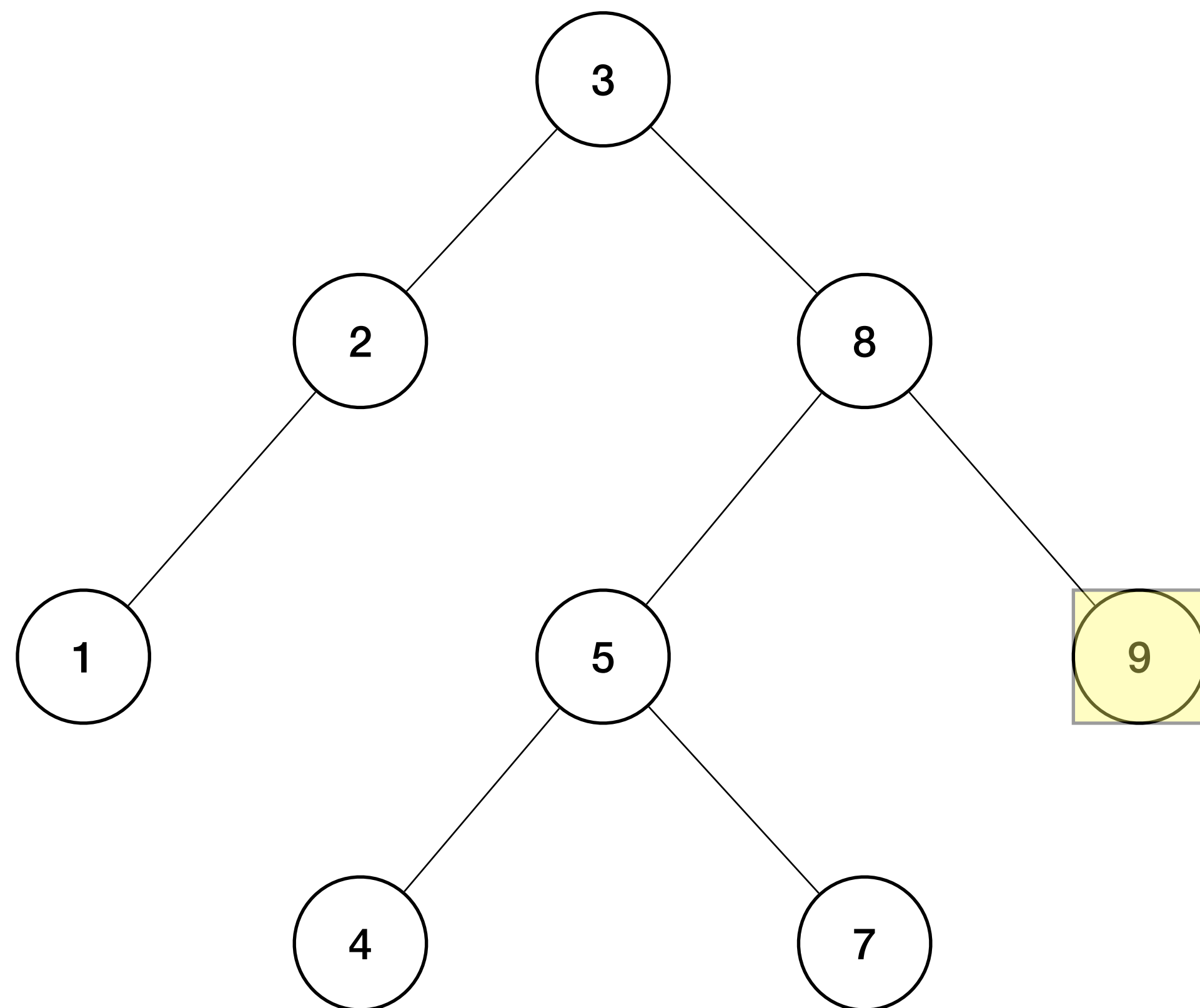
Finding extremals in a BST

Minimum - keep going left

```
node * findmin(node *cursor){
    if (cursor->left==NULL)
        return cursor;
    else
        return findmin(cursor->left);
}
```

Minimum - keep going right

```
node * findmax(node *cursor){
    if (cursor->right==NULL)
        return cursor;
    else
        return findmax(cursor->right);
}
```



```
typedef struct node{
    int data;
    struct node *left;
    struct node *right;
} node;
```

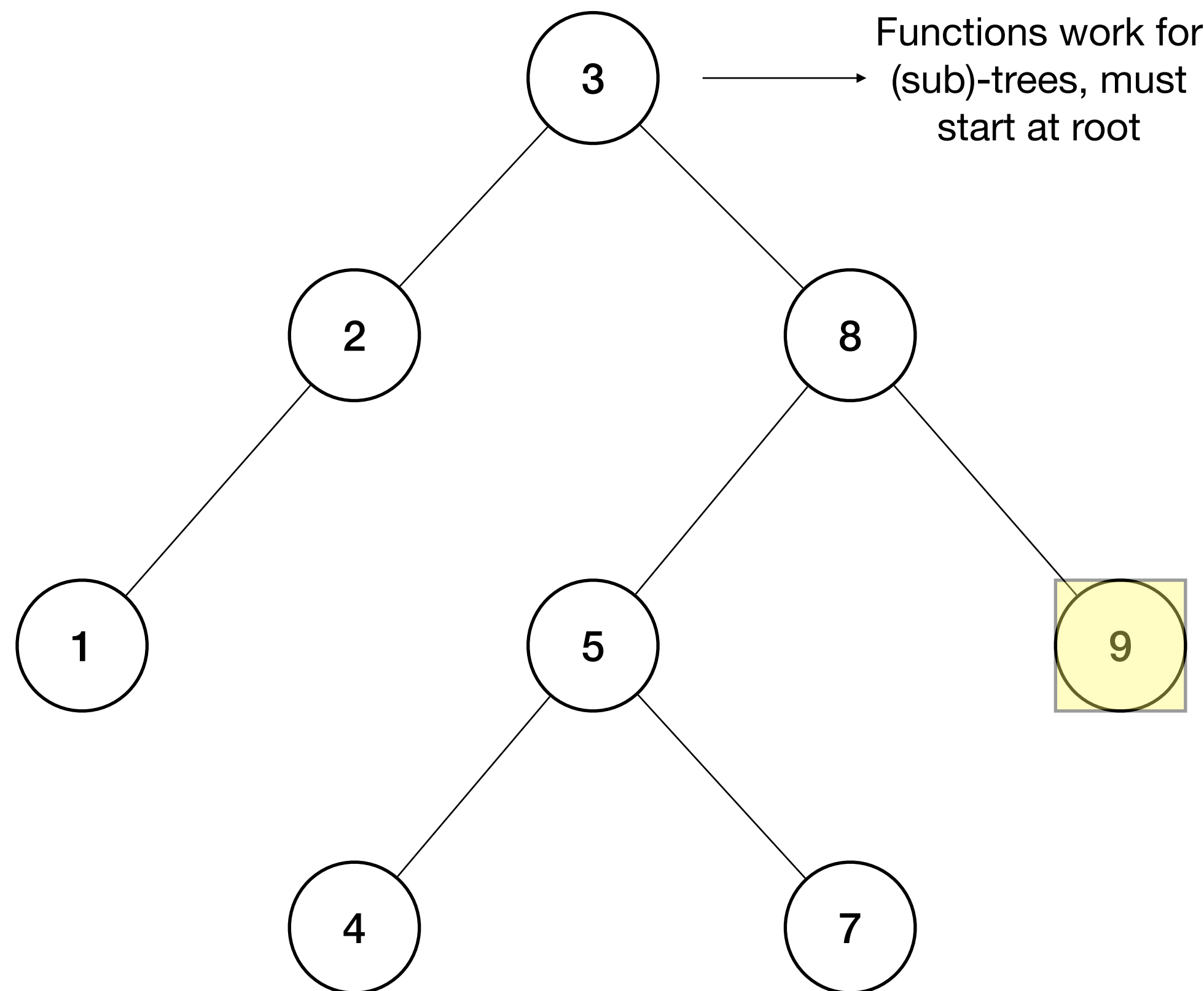
Finding extremals in a BST

Minimum - keep going left

```
node * findmin(node *cursor){
    if (cursor->left==NULL)
        return cursor;
    else
        return findmin(cursor->left);
}
```

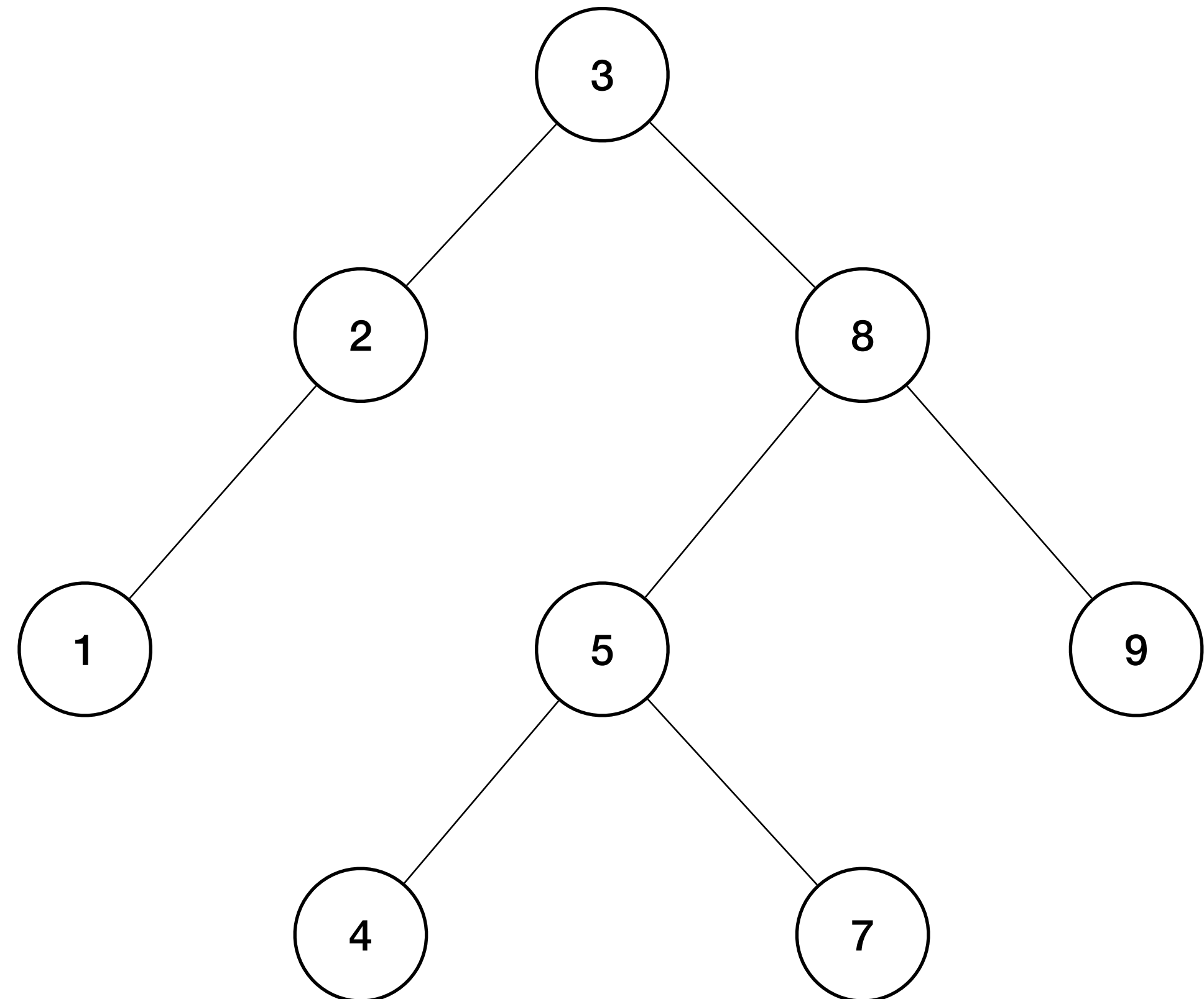
Minimum - keep going right

```
node * findmax(node *cursor){
    if (cursor->right==NULL)
        return cursor;
    else
        return findmax(cursor->right);
}
```



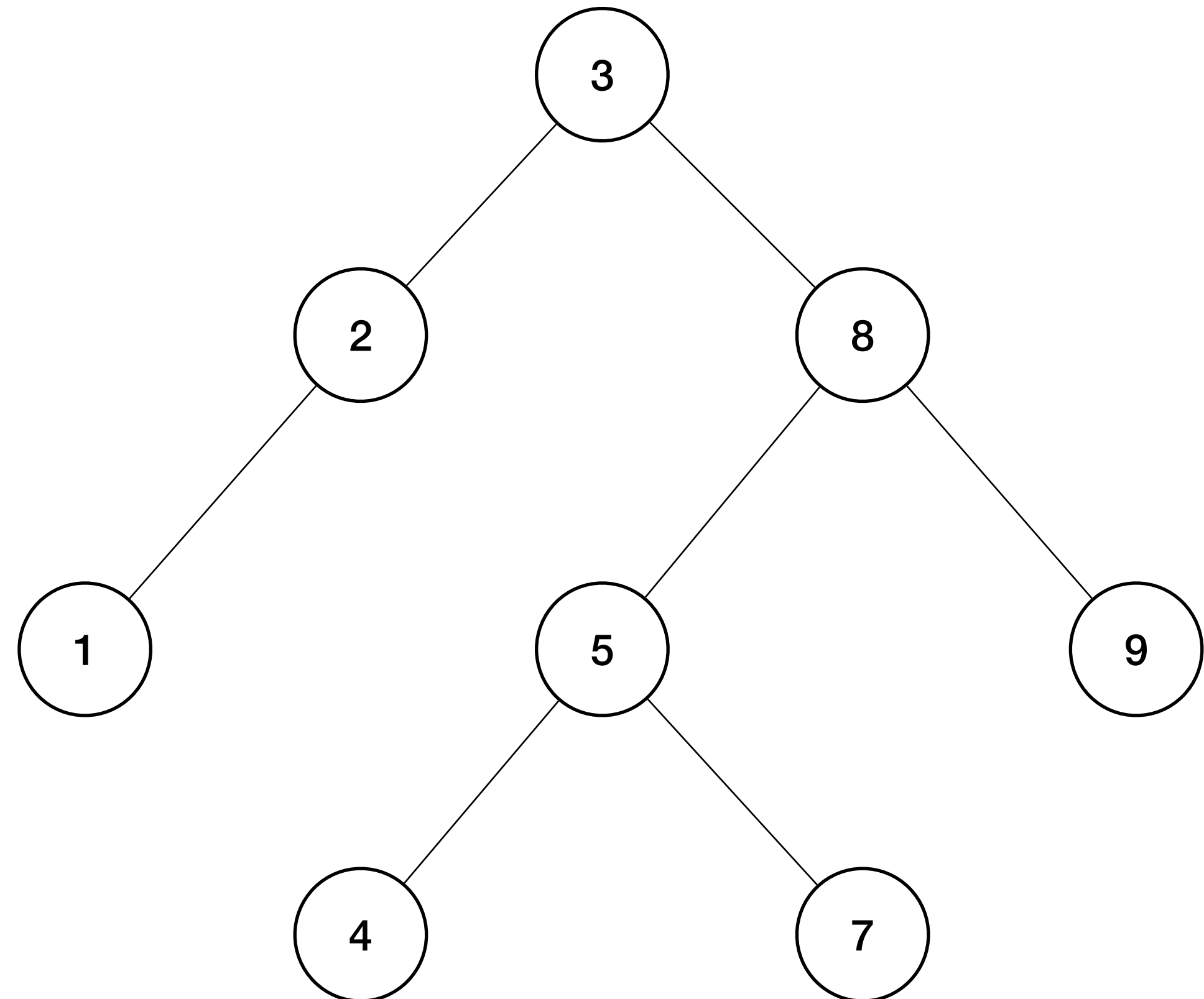
Searching in a BST

```
node * find_elem(node *cursor, int key){  
    if (cursor==NULL) // Key not found  
  
    if (cursor->data == key)  
        // Found key  
    if (cursor->data < key)  
  
else  
  
}
```



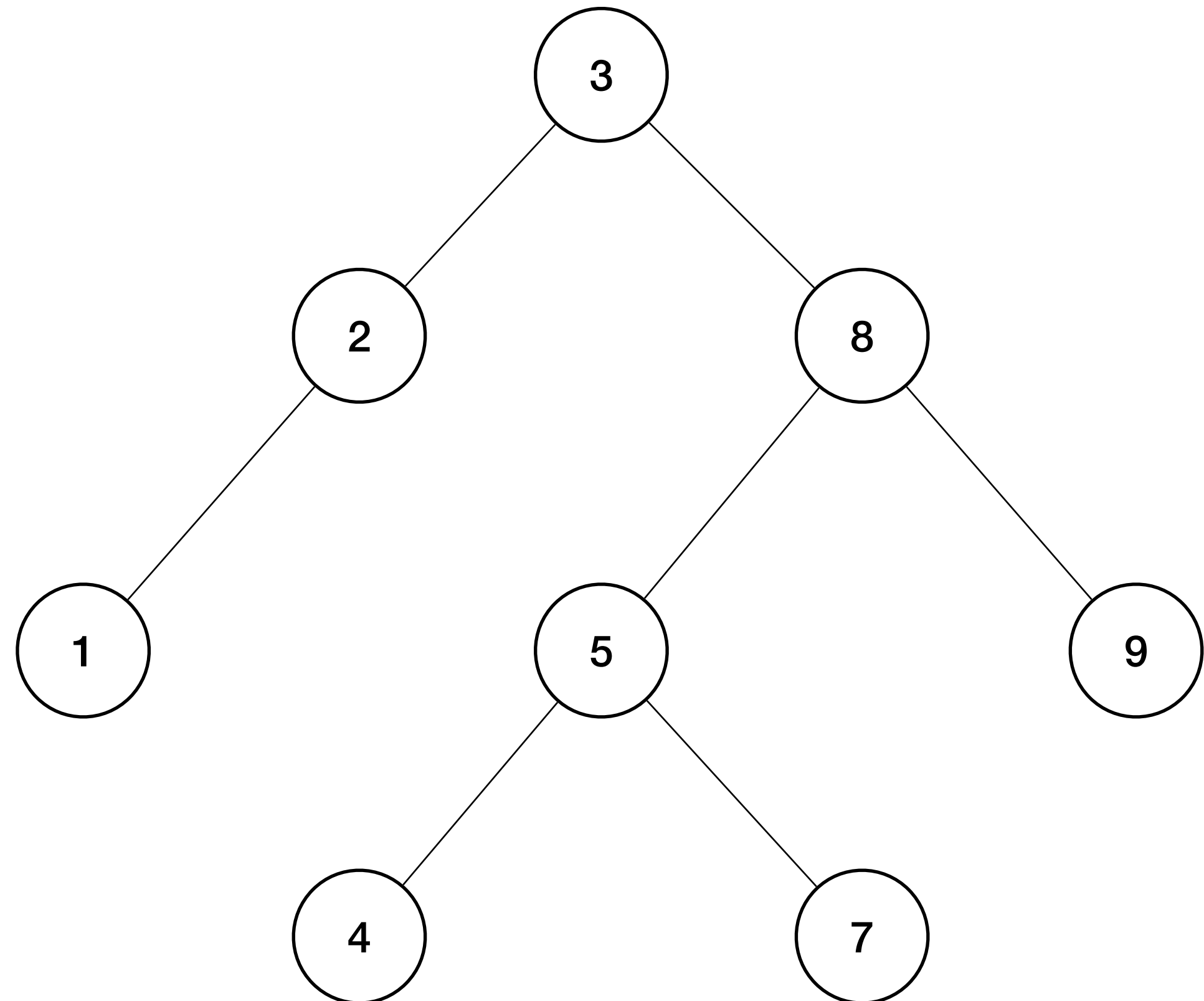
Searching in a BST

```
node * find_elem(node *cursor, int key){  
    if (cursor==NULL) // Key not found  
        return NULL;  
    if (cursor->data == key)  
        // Found key  
    if (cursor->data < key)  
  
else  
  
}
```



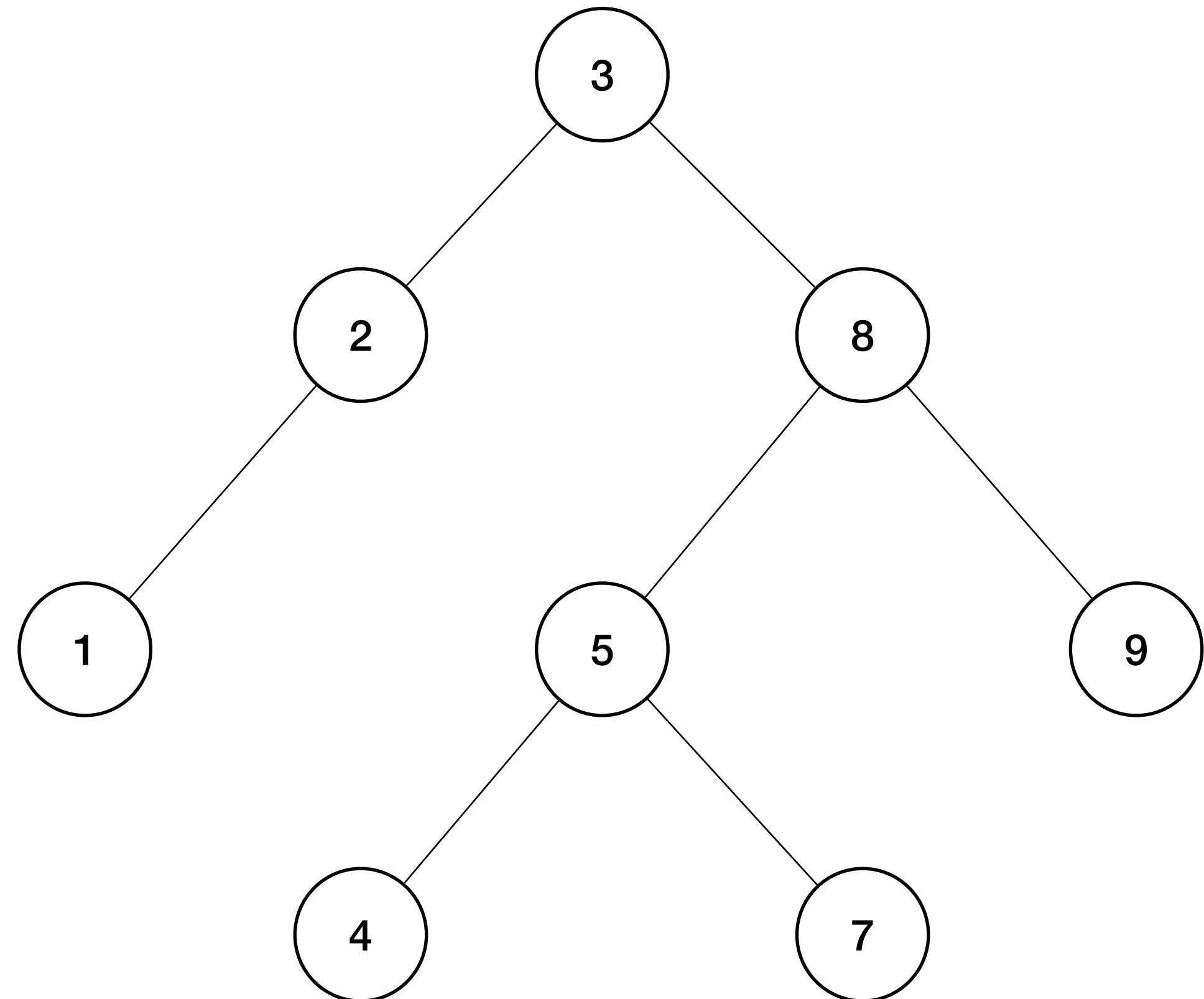
Searching in a BST

```
node * find_elem(node *cursor, int key){  
    if (cursor==NULL) // Key not found  
        return NULL;  
    if (cursor->data == key)  
        return cursor; // Found key  
    if (cursor->data < key)  
  
else  
  
}
```



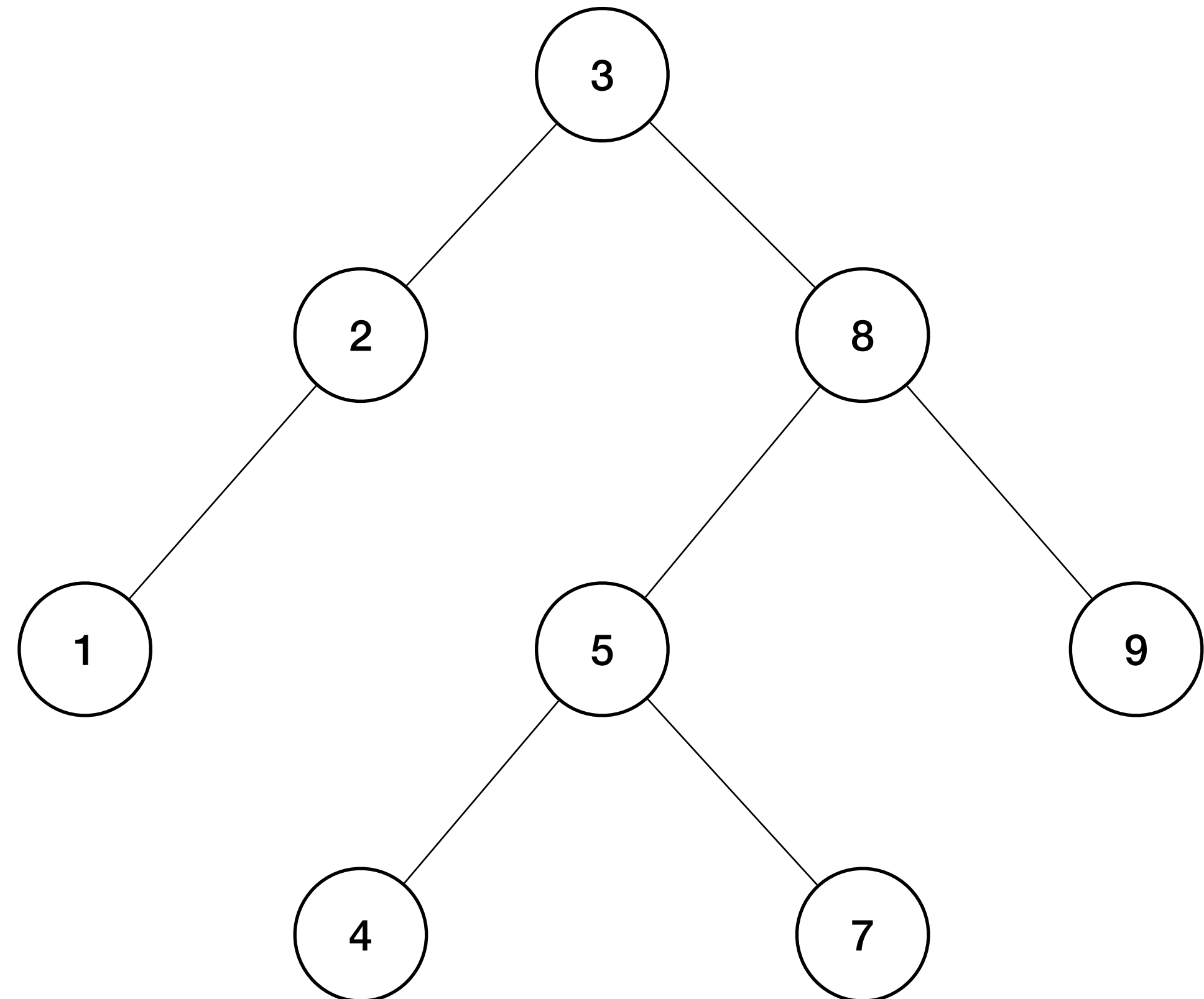
Searching in a BST

```
node * find_elem(node *cursor, int key){  
    if (cursor==NULL) // Key not found  
        return NULL;  
    if (cursor->data == key)  
        return cursor; // Found key  
    if (cursor->data < key)  
        // Go right  
        return find_elem(cursor->right, key);  
    else  
}
```



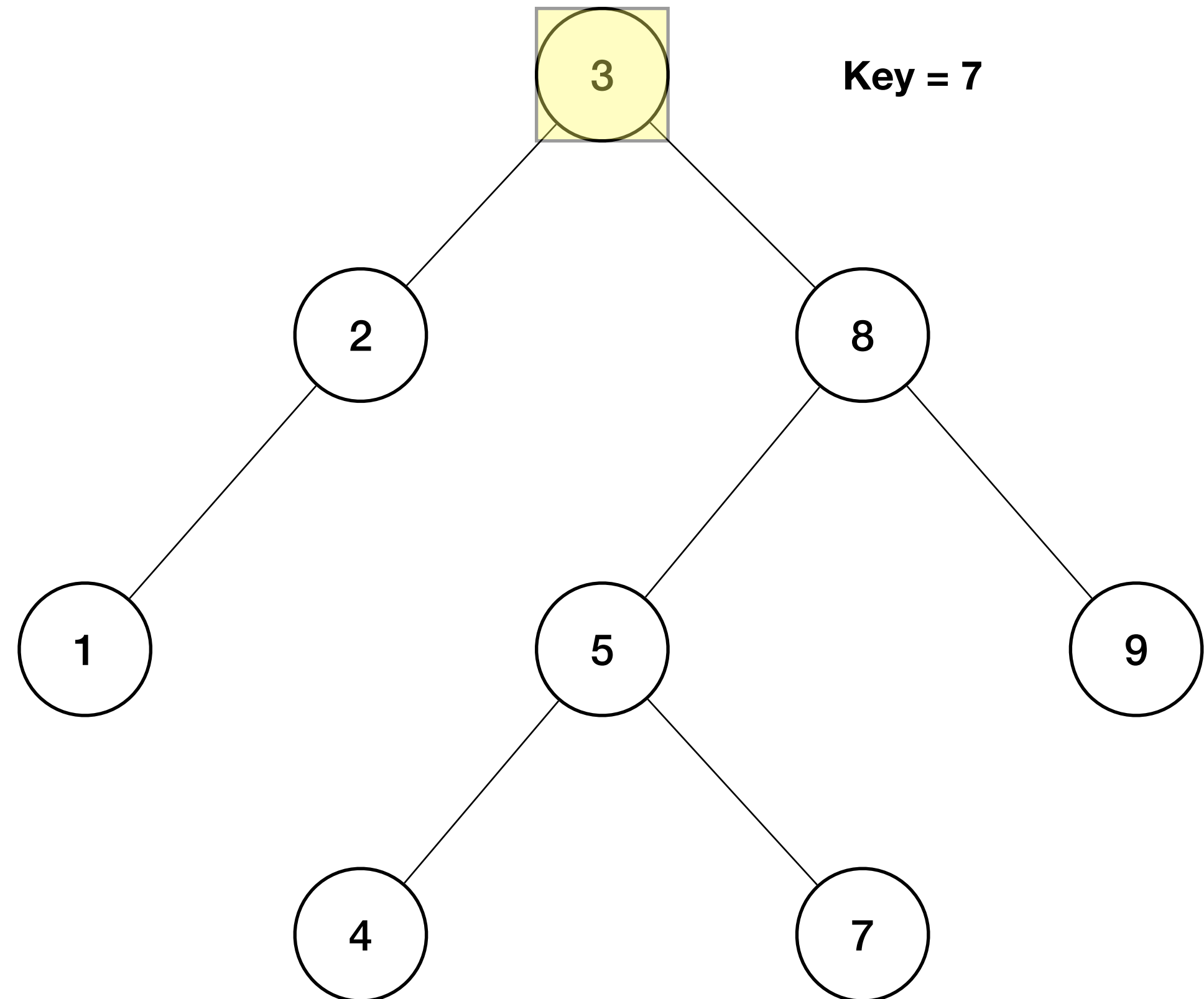
Searching in a BST

```
node * find_elem(node *cursor, int key){
    if (cursor==NULL) // Key not found
        return NULL;
    if (cursor->data == key)
        return cursor; // Found key
    if (cursor->data < key)
        // Go right
        return find_elem(cursor->right, key);
    else
        // Go left
        return find_elem(cursor->left, key);
}
```



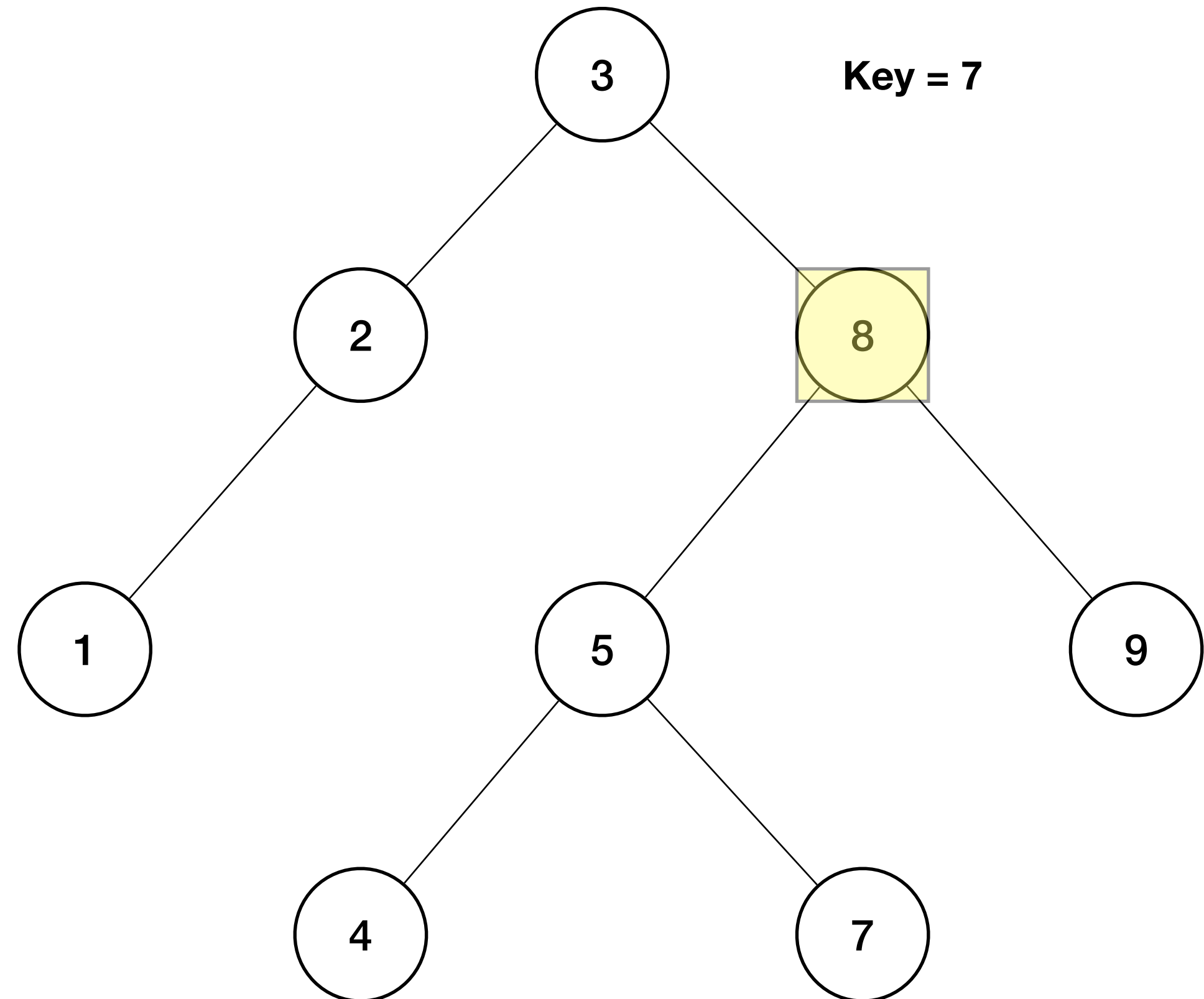
Searching in a BST

```
node * find_elem(node *cursor, int key){
    if (cursor==NULL) // Key not found
        return NULL;
    if (cursor->data == key)
        return cursor; // Found key
    if (cursor->data < key)
        // Go right
        return find_elem(cursor->right, key);
    else
        // Go left
        return find_elem(cursor->left, key);
}
```



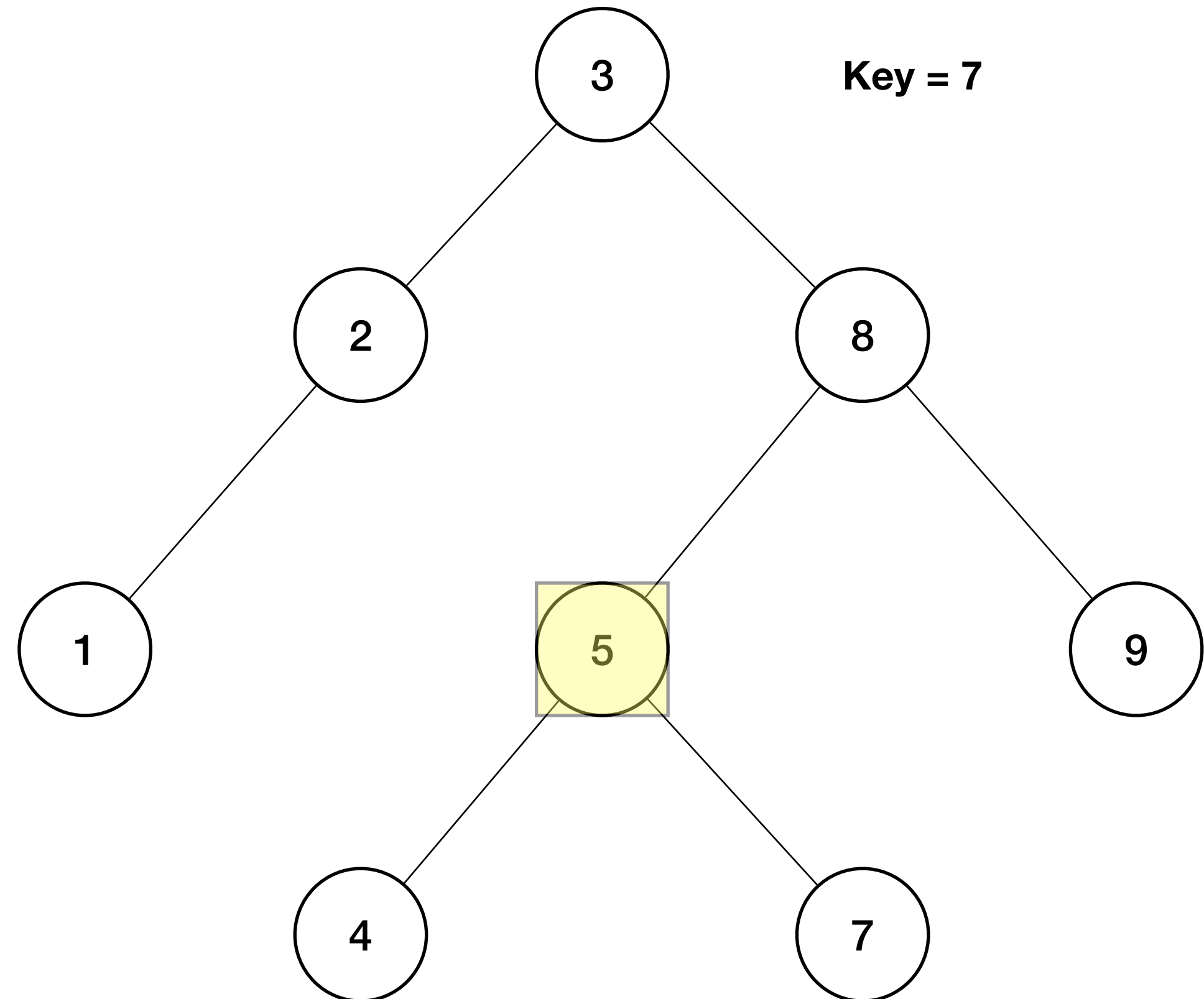
Searching in a BST

```
node * find_elem(node *cursor, int key){  
    if (cursor==NULL) // Key not found  
        return NULL;  
    if (cursor->data == key)  
        return cursor; // Found key  
    if (cursor->data < key)  
        // Go right  
        return find_elem(cursor->right, key);  
    else  
        // Go left  
        return find_elem(cursor->left, key);  
}
```



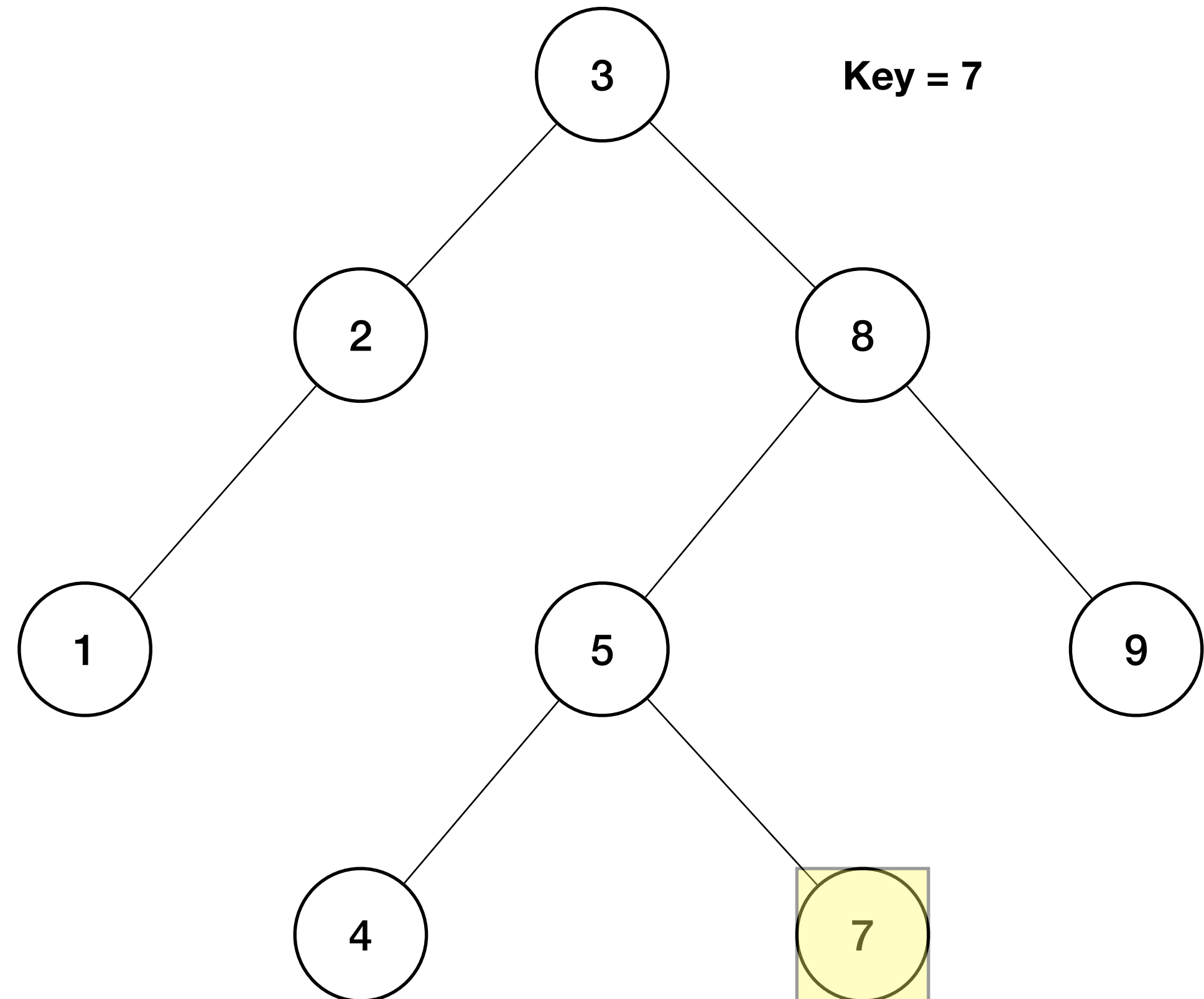
Searching in a BST

```
node * find_elem(node *cursor, int key){
    if (cursor==NULL) // Key not found
        return NULL;
    if (cursor->data == key)
        return cursor; // Found key
    if (cursor->data < key)
        // Go right
        return find_elem(cursor->right, key);
    else
        // Go left
        return find_elem(cursor->left, key);
}
```

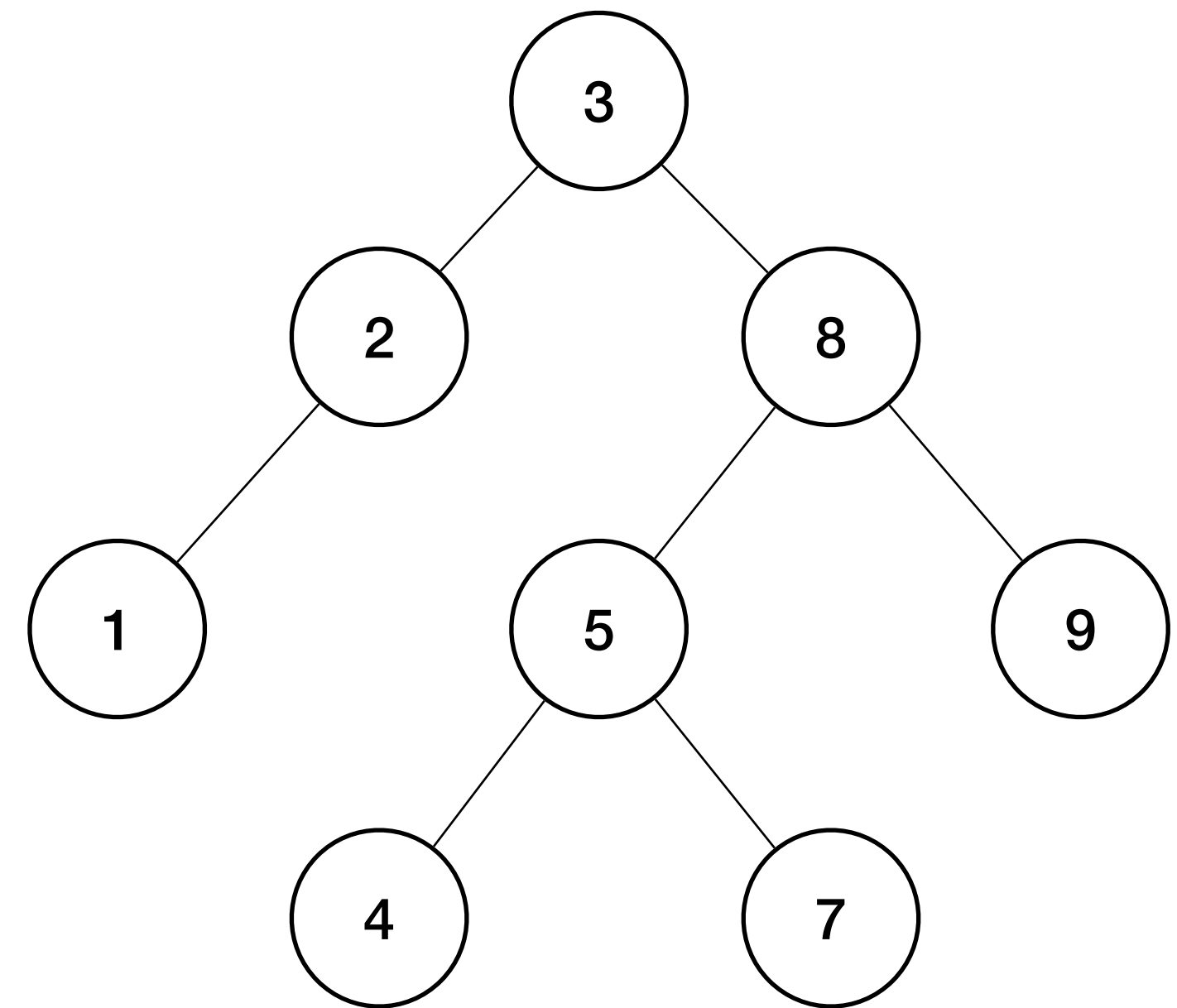


Searching in a BST

```
node * find_elem(node *cursor, int key){  
    if (cursor==NULL) // Key not found  
        return NULL;  
    if (cursor->data == key)  
        return cursor; // Found key  
    if (cursor->data < key)  
        // Go right  
        return find_elem(cursor->right, key);  
    else  
        // Go left  
        return find_elem(cursor->left, key);  
}
```

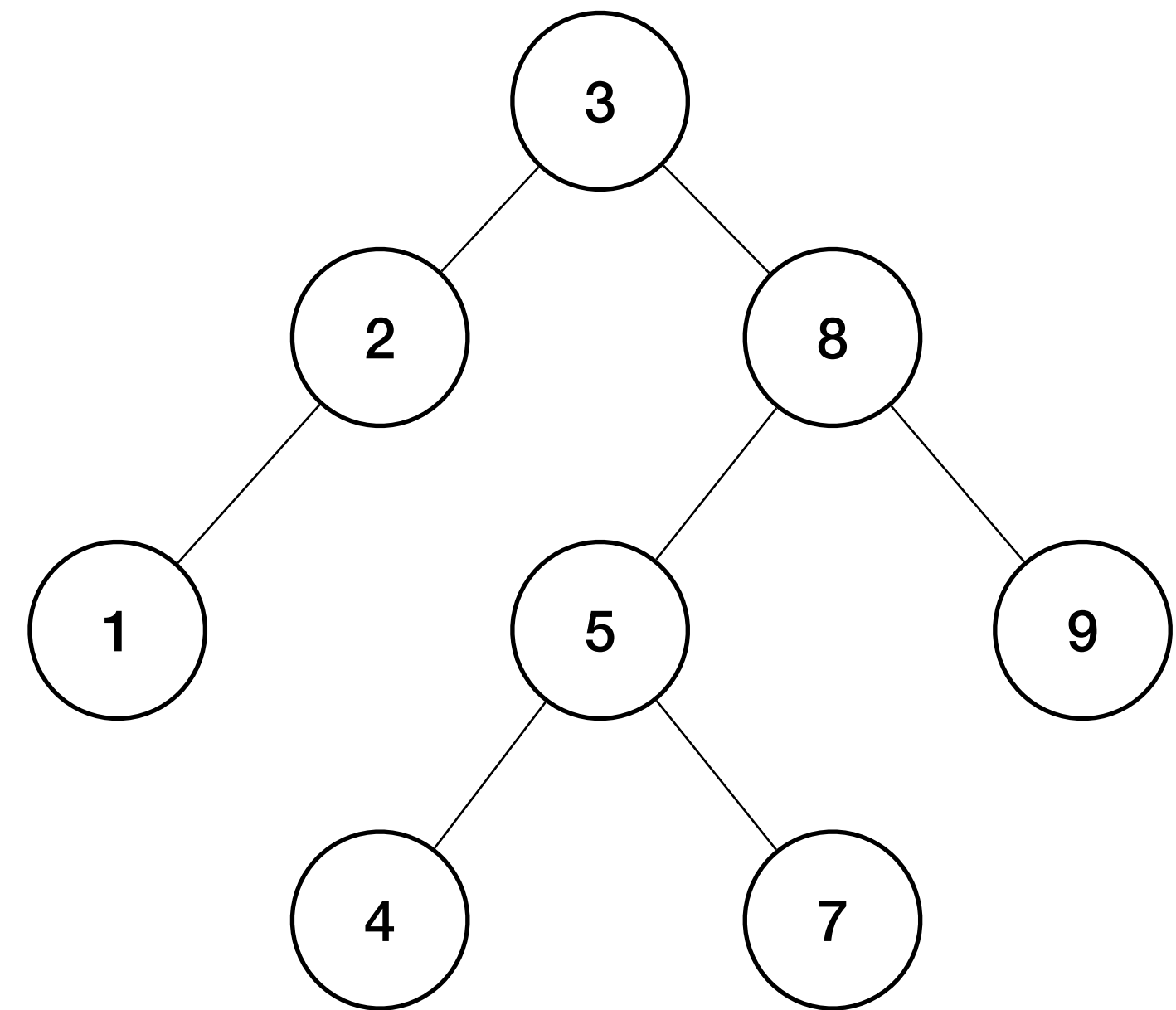


Insertion in a BST



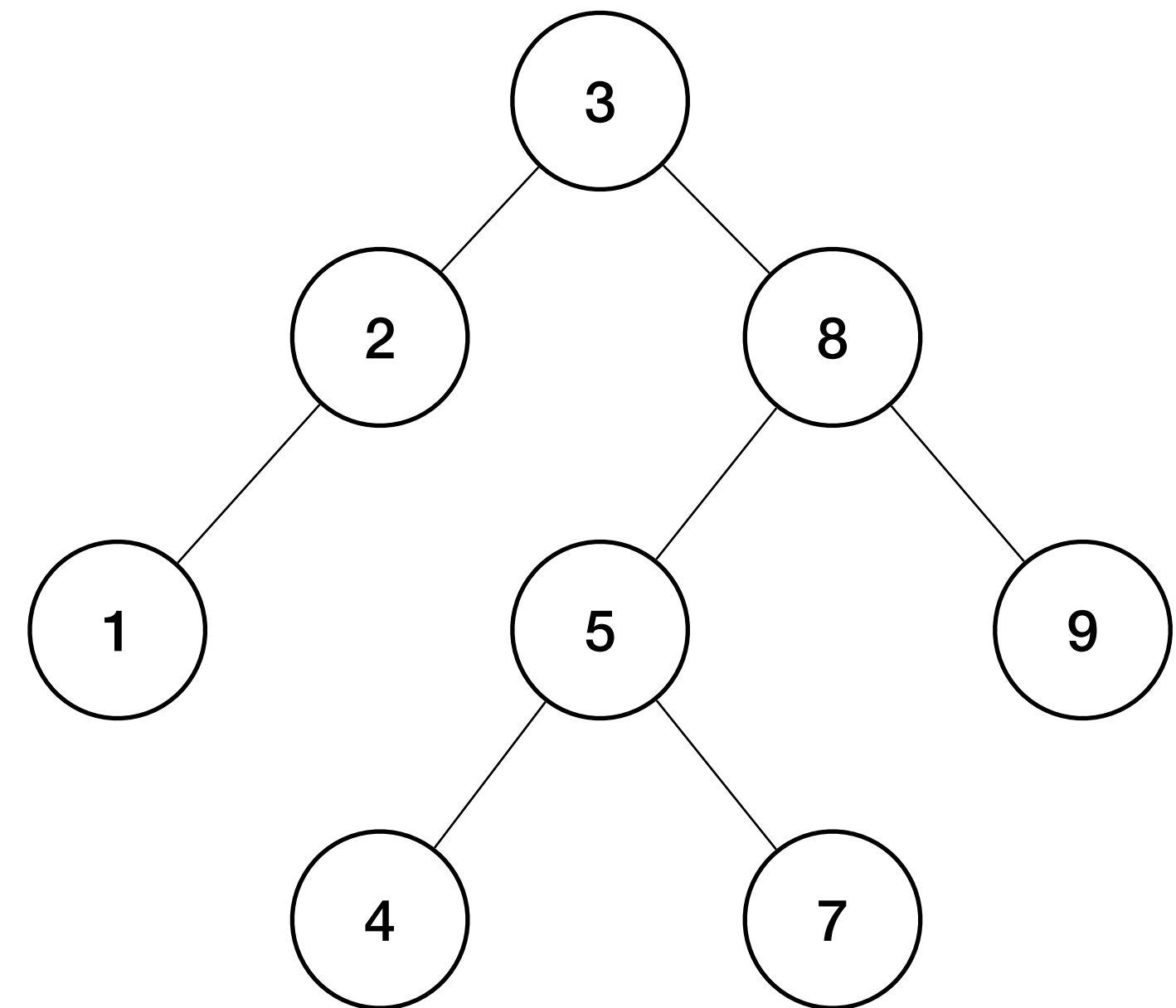
Insertion in a BST

- Insertions need to preserve the BST property



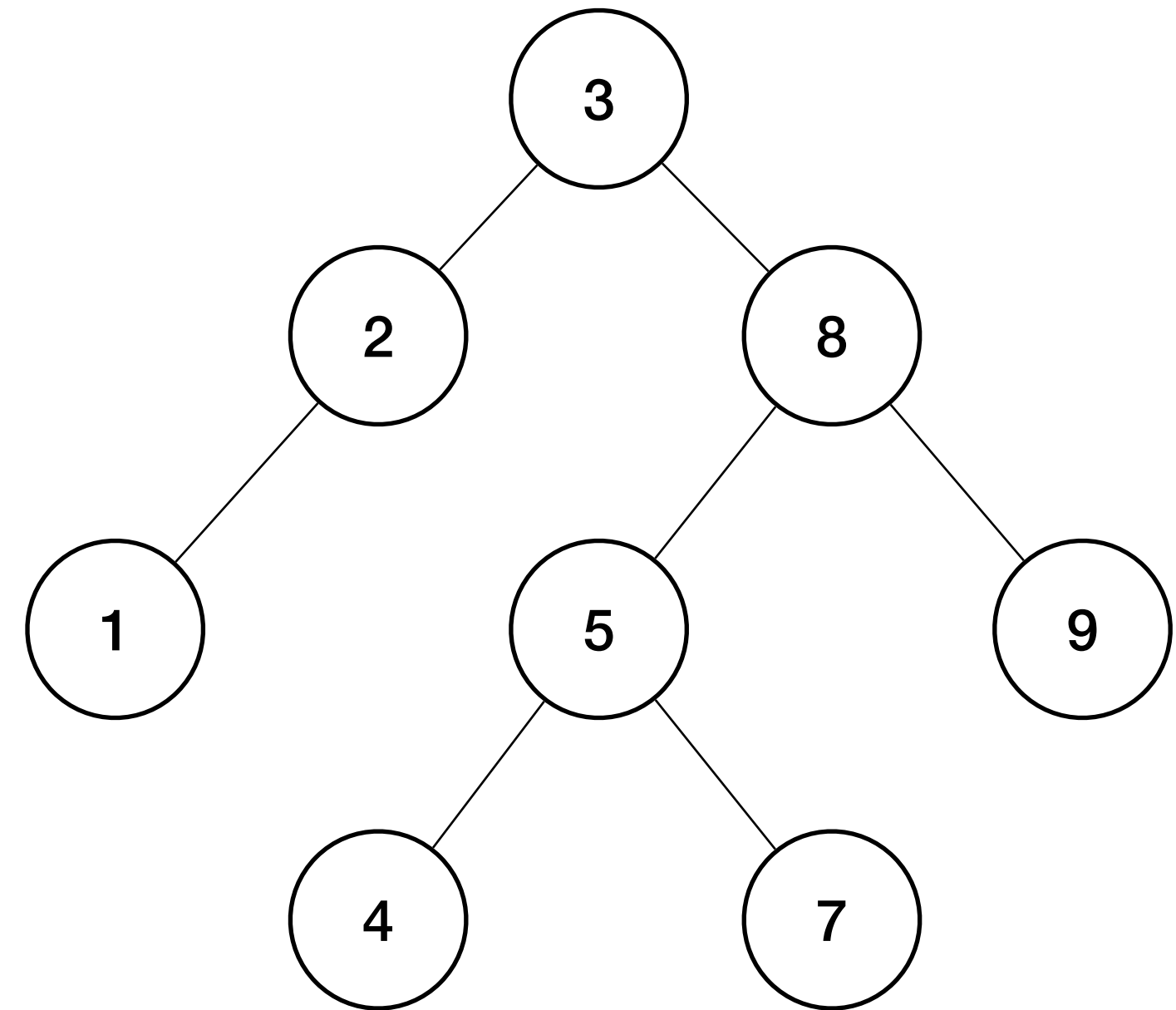
Insertion in a BST

- Insertions need to preserve the BST property
- Add new nodes only as leaf nodes



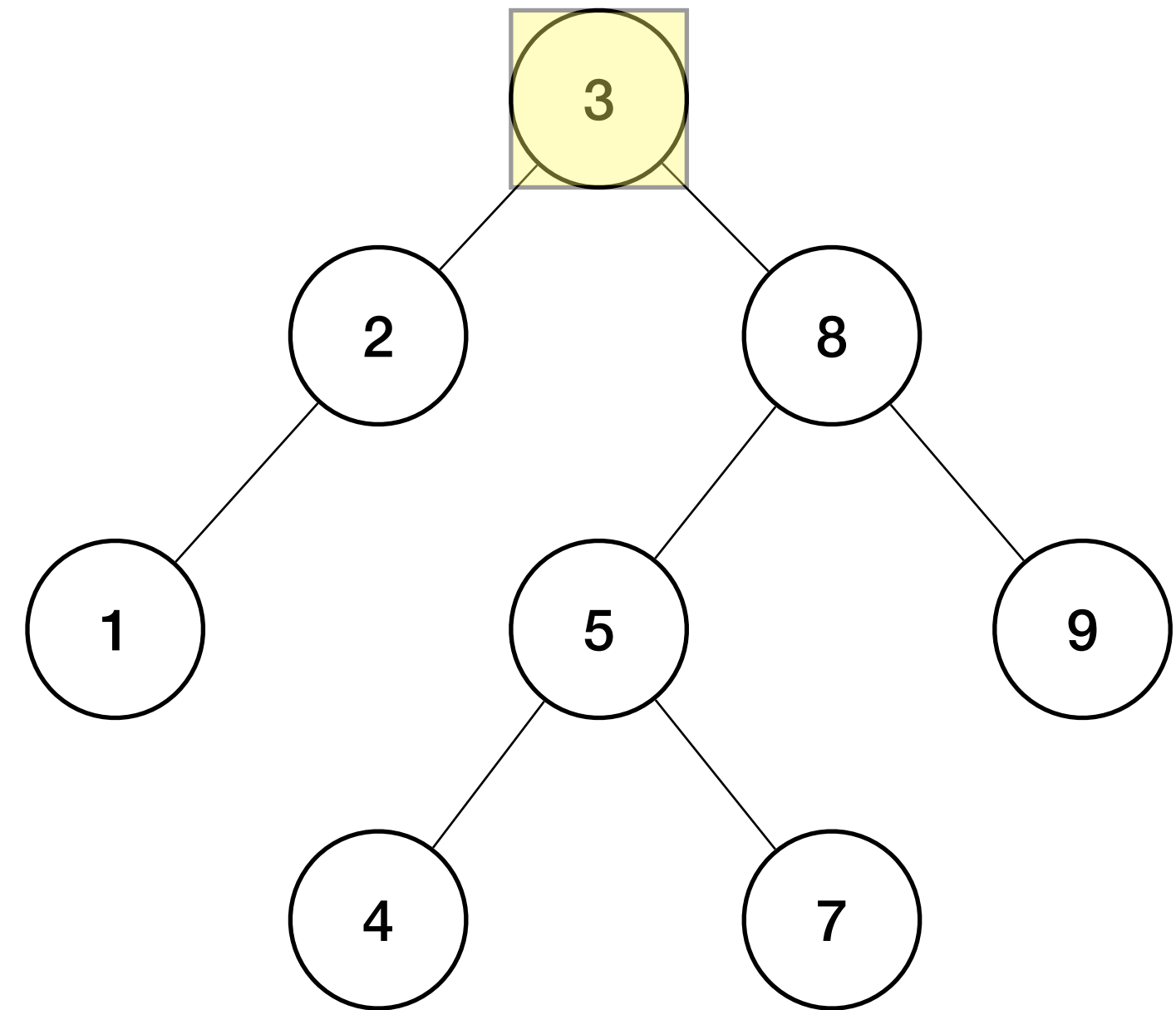
Insertion in a BST

- Insertions need to preserve the BST property
- Add new nodes only as leaf nodes
- Consider inserting 6 in the BST on the right ...



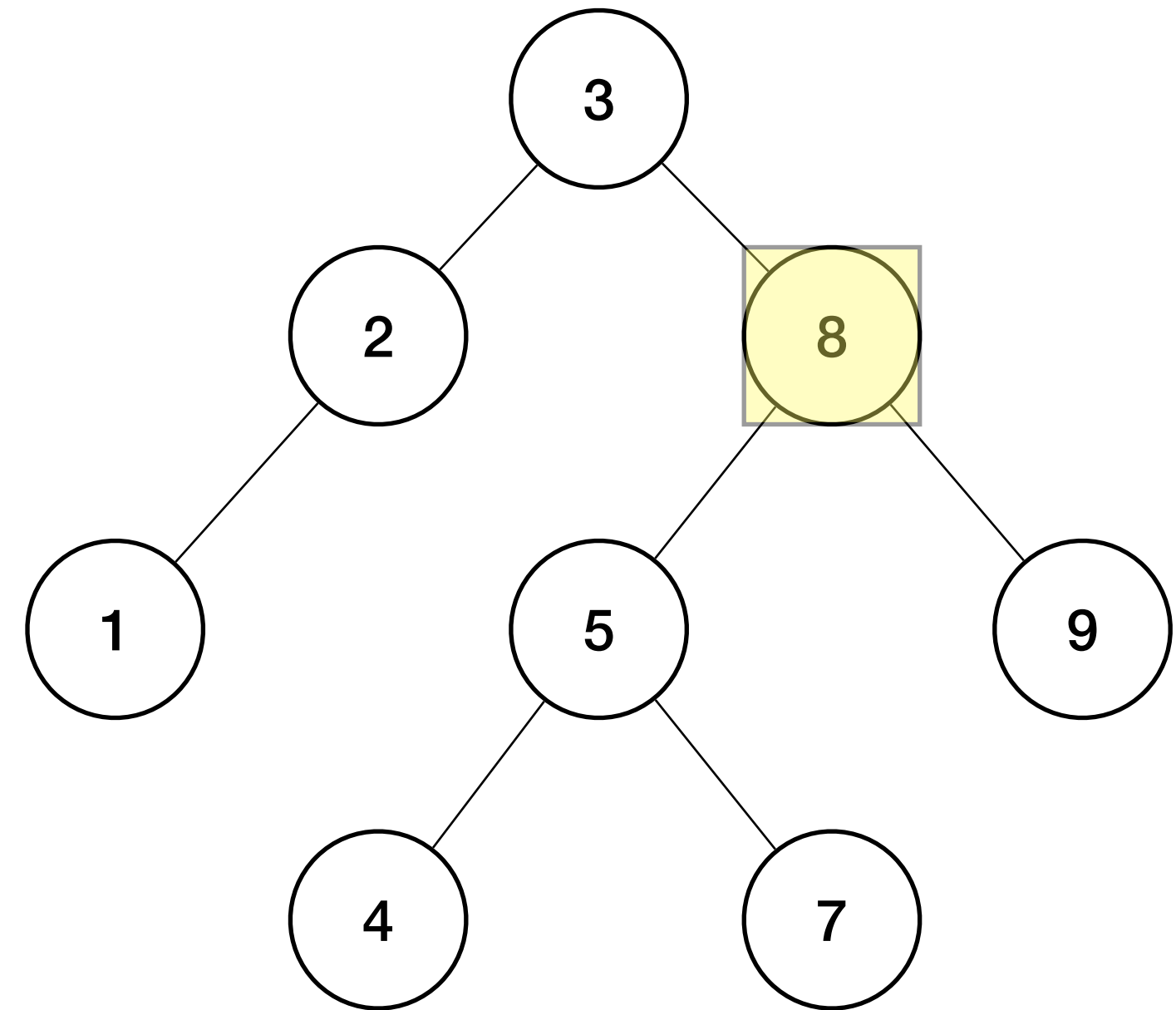
Insertion in a BST

- Insertions need to preserve the BST property
- Add new nodes only as leaf nodes
- Consider inserting 6 in the BST on the right ...



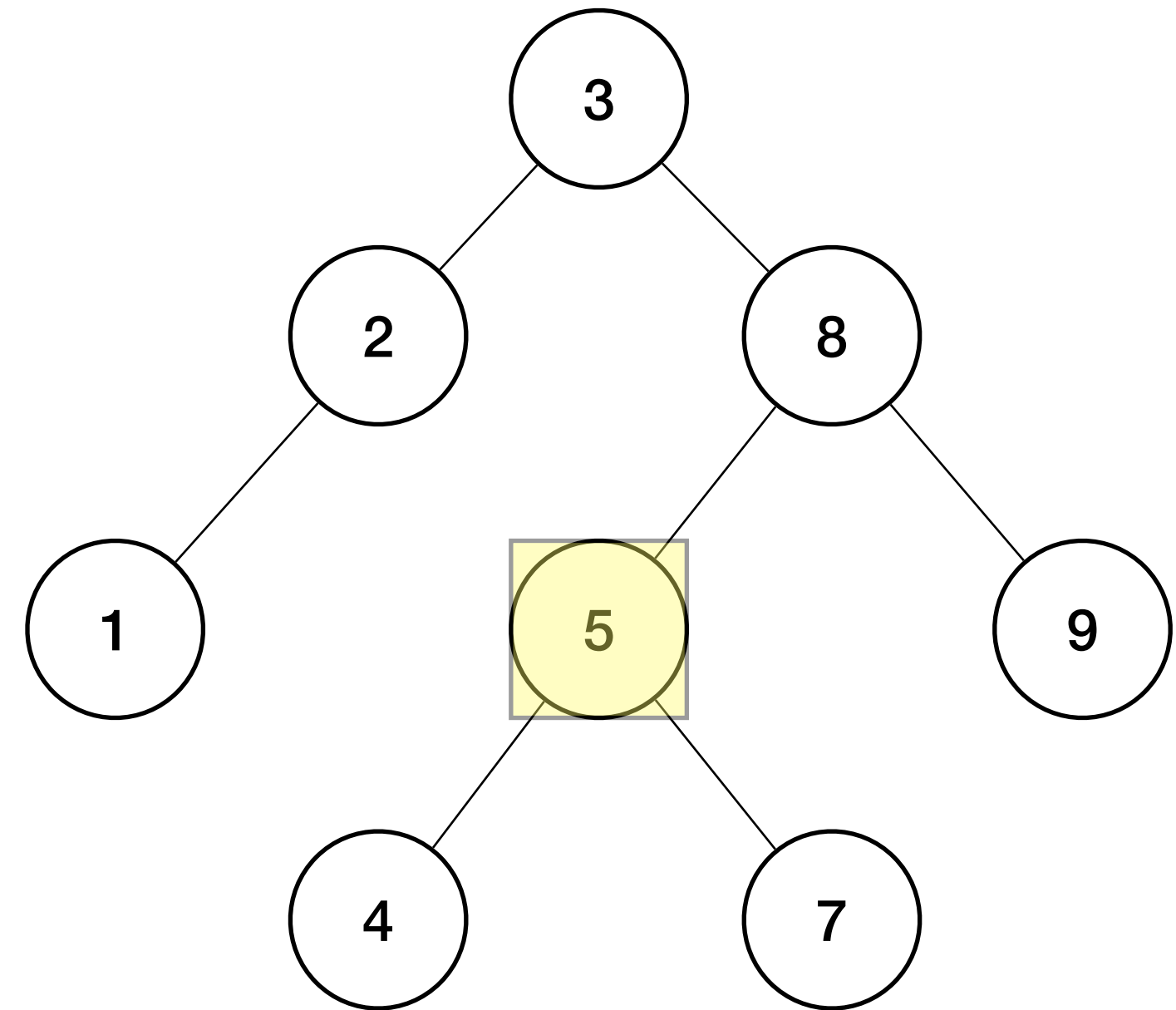
Insertion in a BST

- Insertions need to preserve the BST property
- Add new nodes only as leaf nodes
- Consider inserting 6 in the BST on the right ...



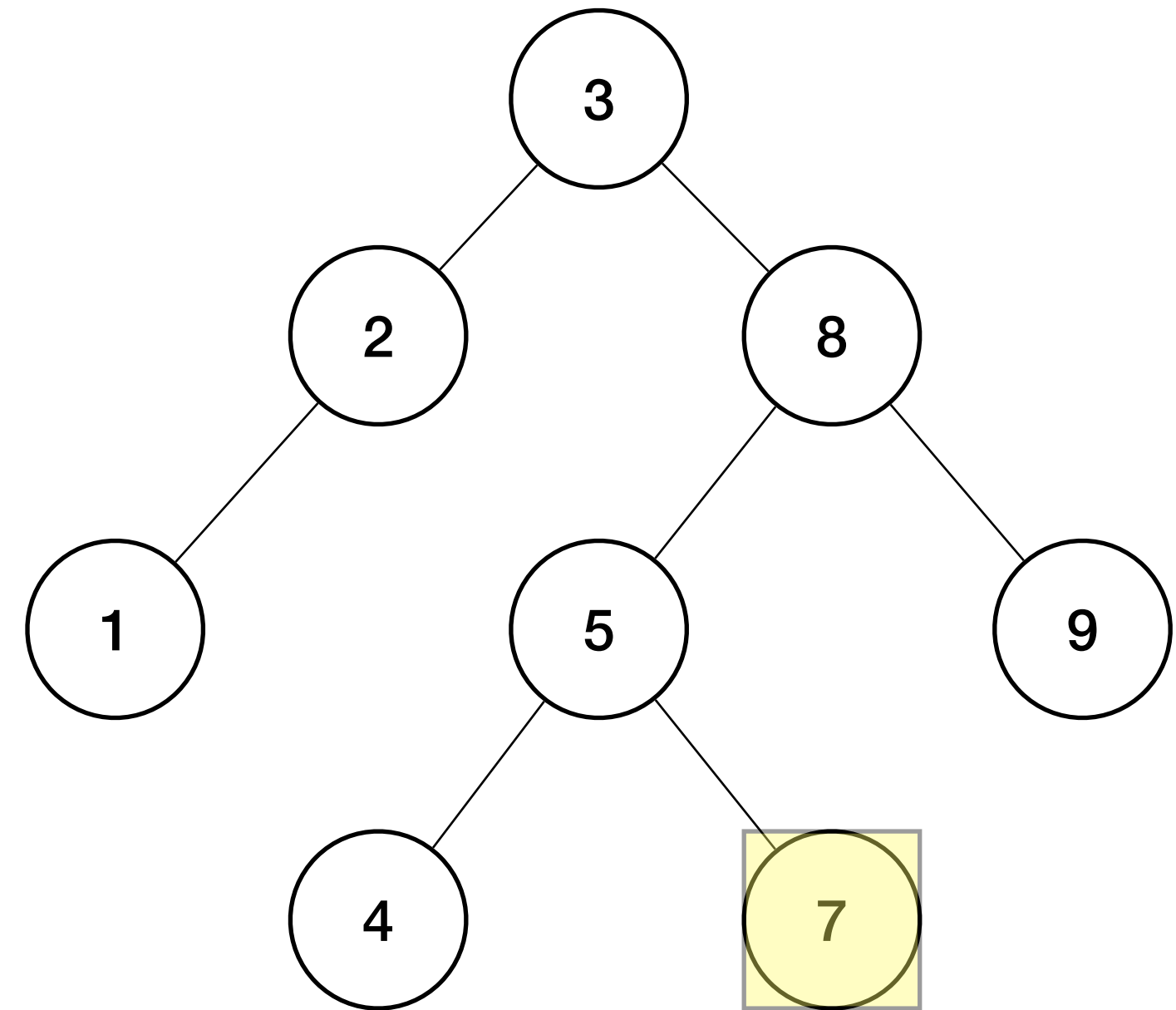
Insertion in a BST

- Insertions need to preserve the BST property
- Add new nodes only as leaf nodes
- Consider inserting 6 in the BST on the right ...



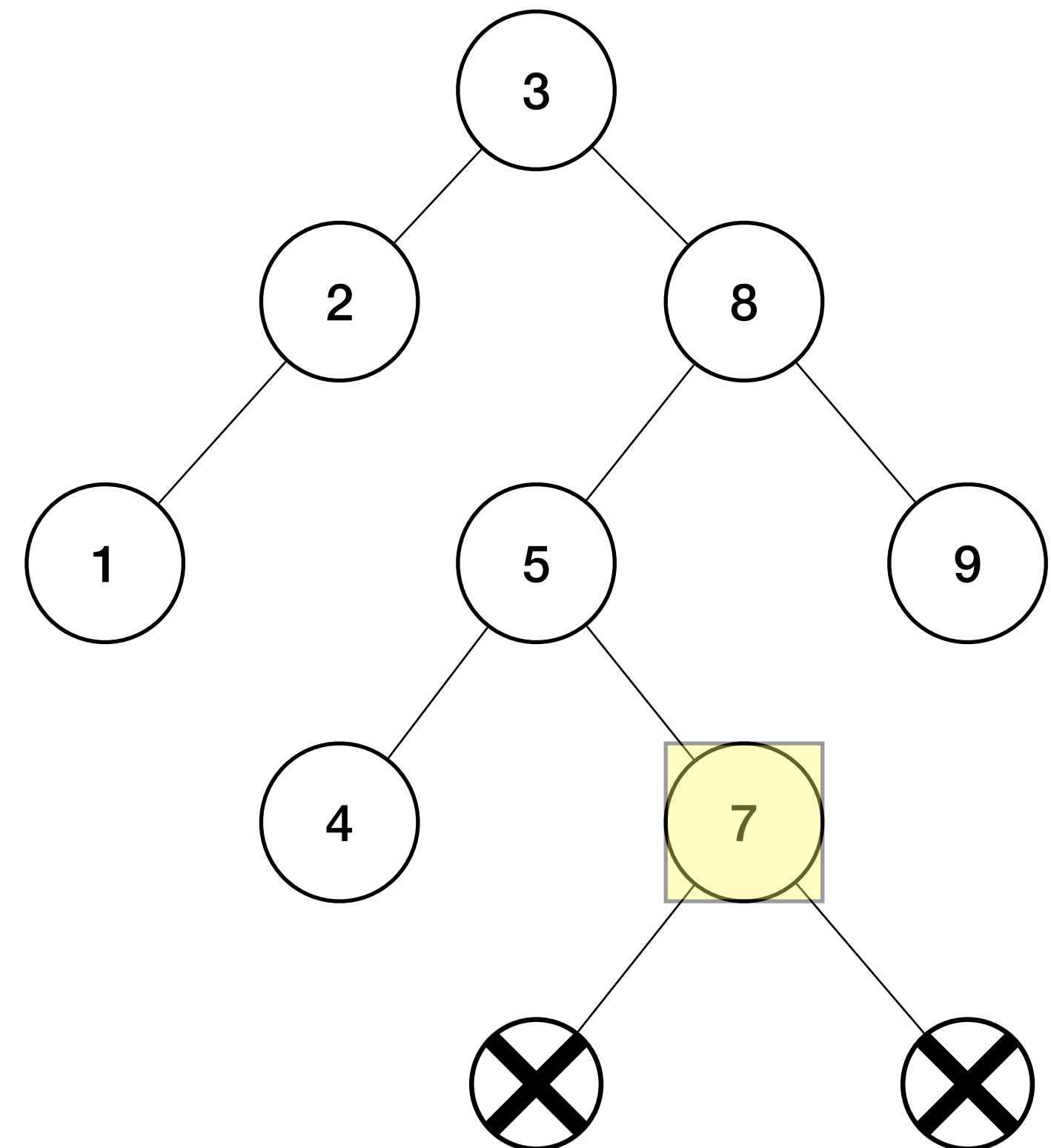
Insertion in a BST

- Insertions need to preserve the BST property
- Add new nodes only as leaf nodes
- Consider inserting 6 in the BST on the right ...



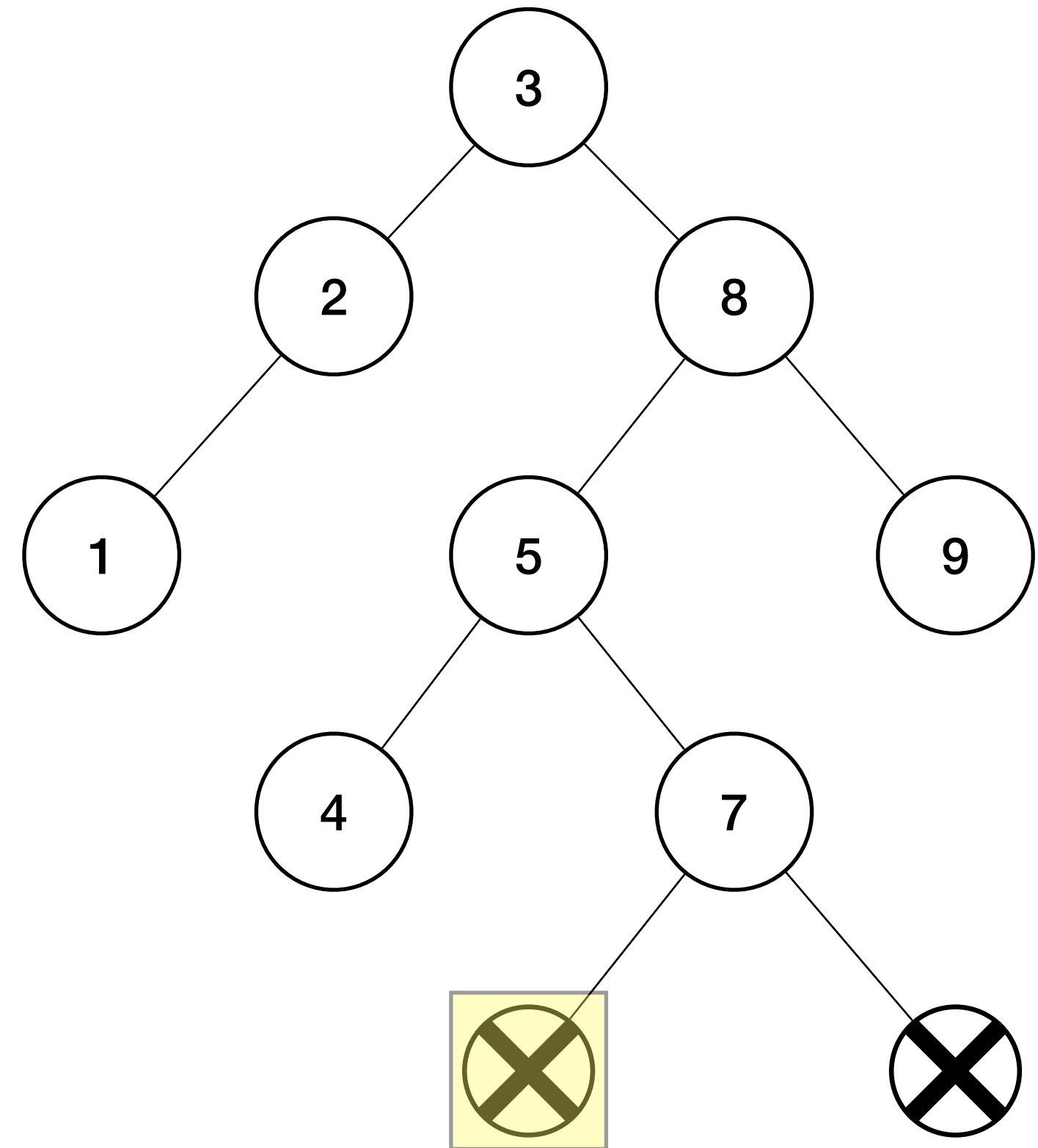
Insertion in a BST

- Insertions need to preserve the BST property
- Add new nodes only as leaf nodes
- Consider inserting 6 in the BST on the right ...



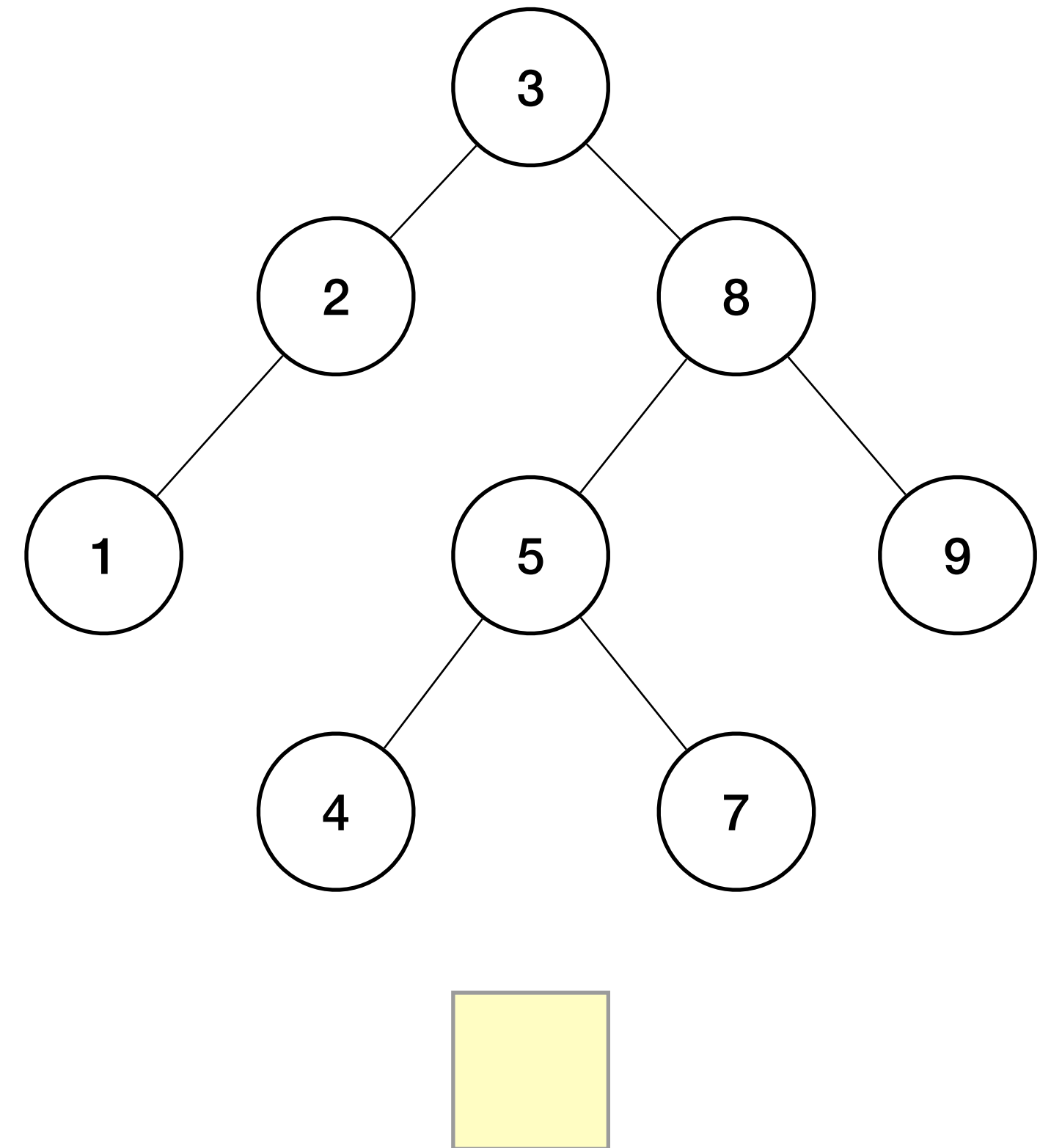
Insertion in a BST

- Insertions need to preserve the BST property
- Add new nodes only as leaf nodes
- Consider inserting 6 in the BST on the right ...



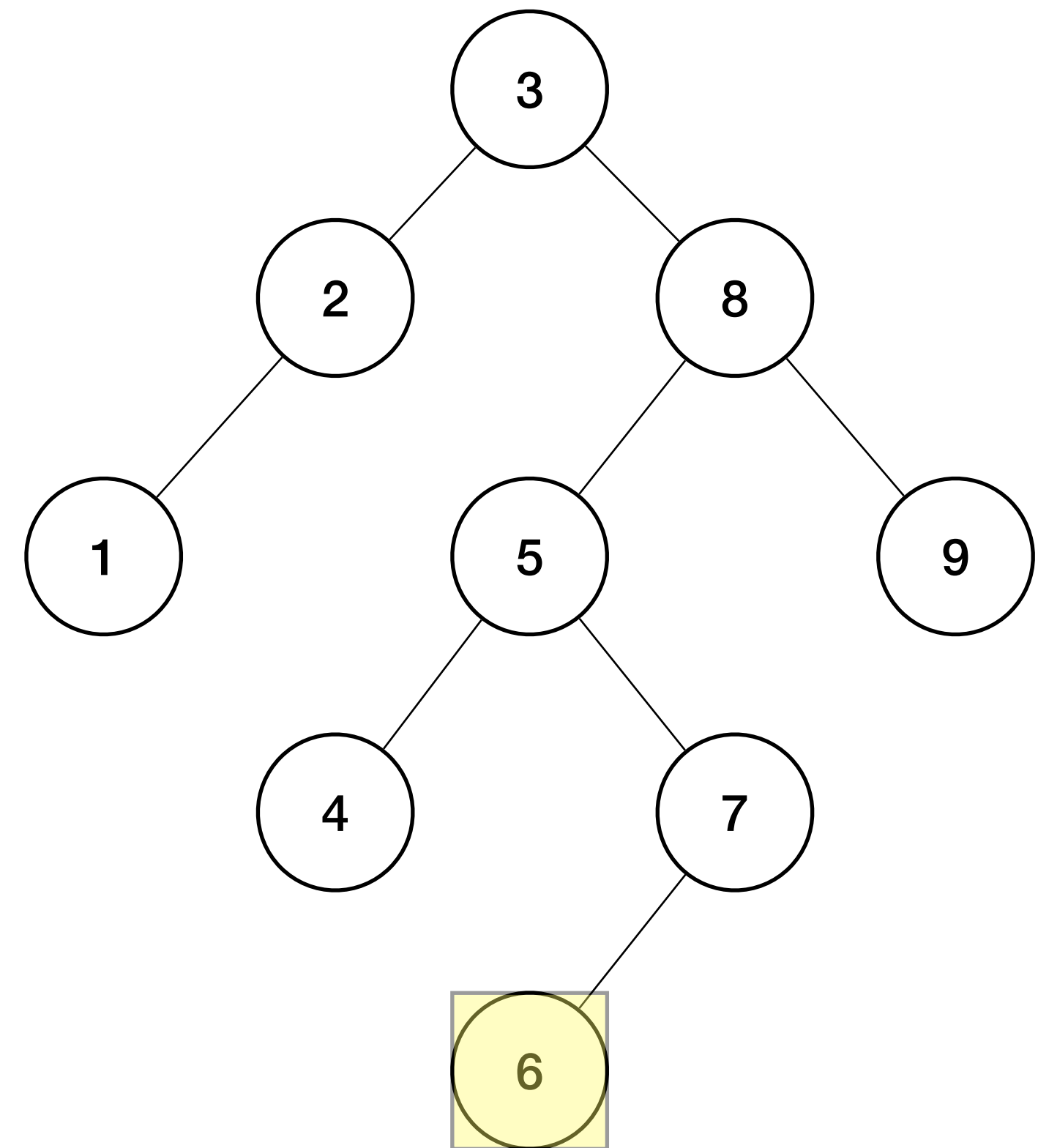
Insertion in a BST

- Insertions need to preserve the BST property
- Add new nodes only as leaf nodes
- Consider inserting 6 in the BST on the right ...



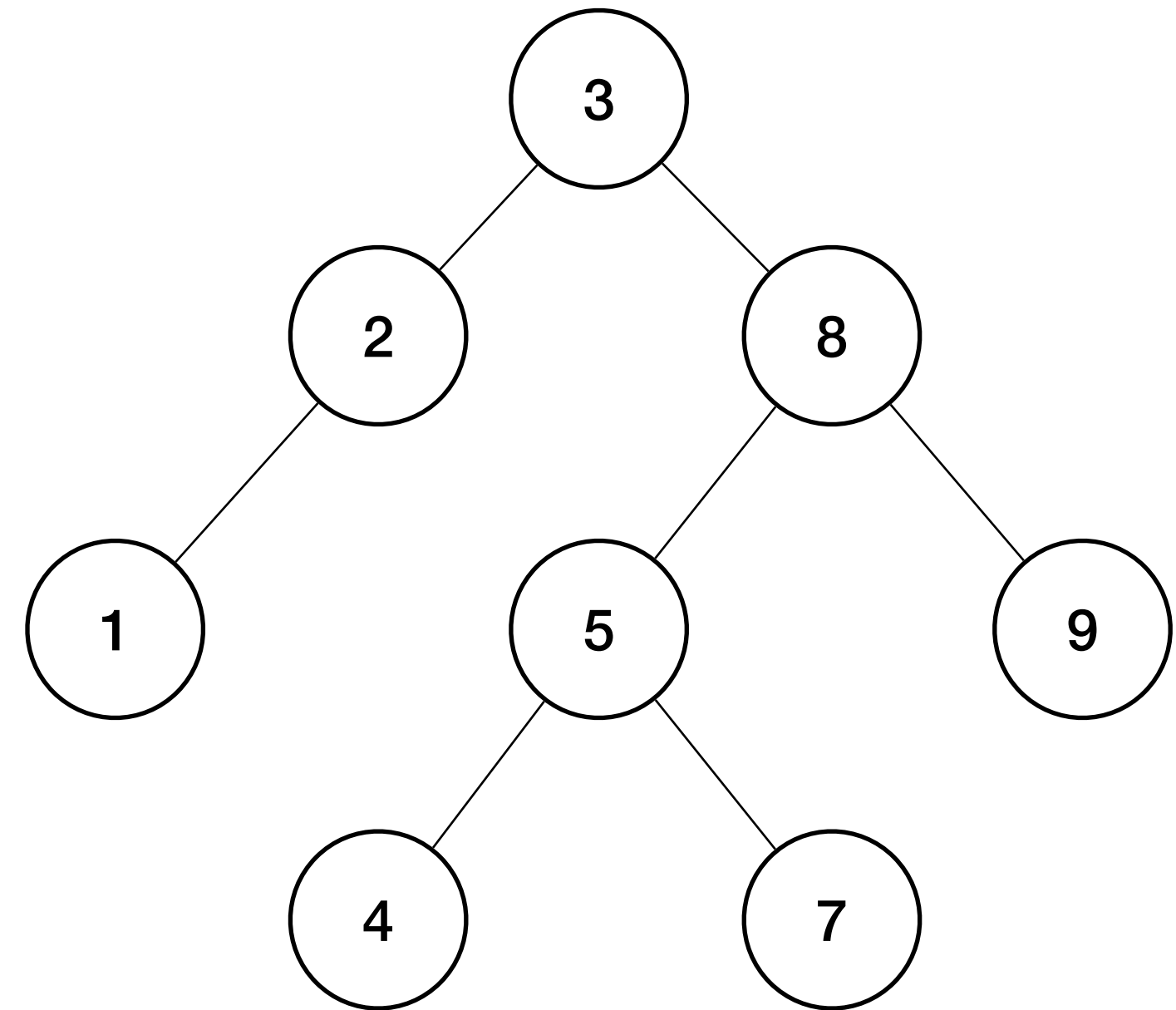
Insertion in a BST

- Insertions need to preserve the BST property
- Add new nodes only as leaf nodes
- Consider inserting 6 in the BST on the right ...



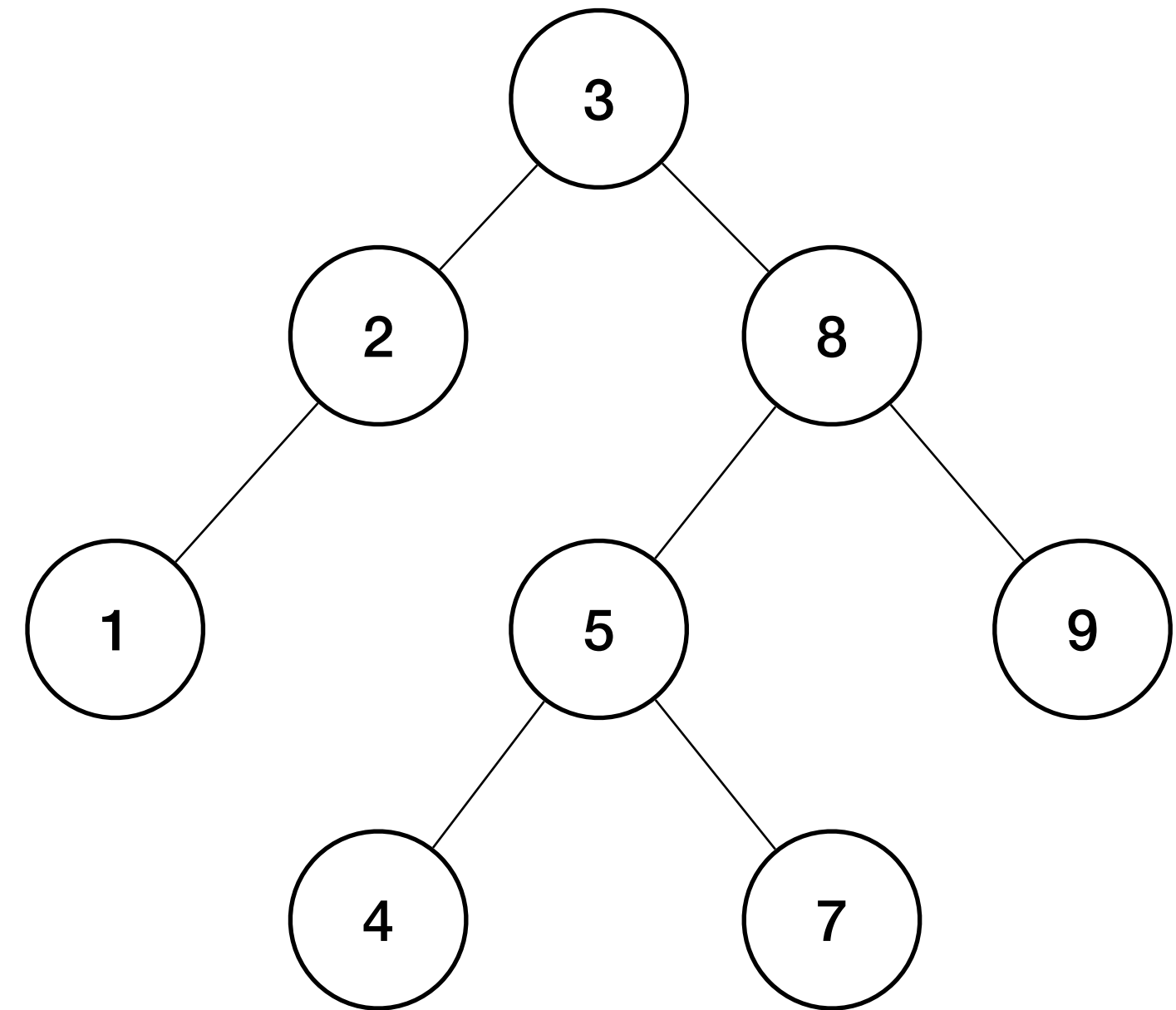
Insertion in a BST

```
node * insert(node *cursor, int data){  
  if (cursor==NULL)  
    return newNode(data);  
  else{  
    if (data < cursor->data)  
      cursor->left =  
    else  
      cursor->right =  
    return cursor;  
  }  
}
```



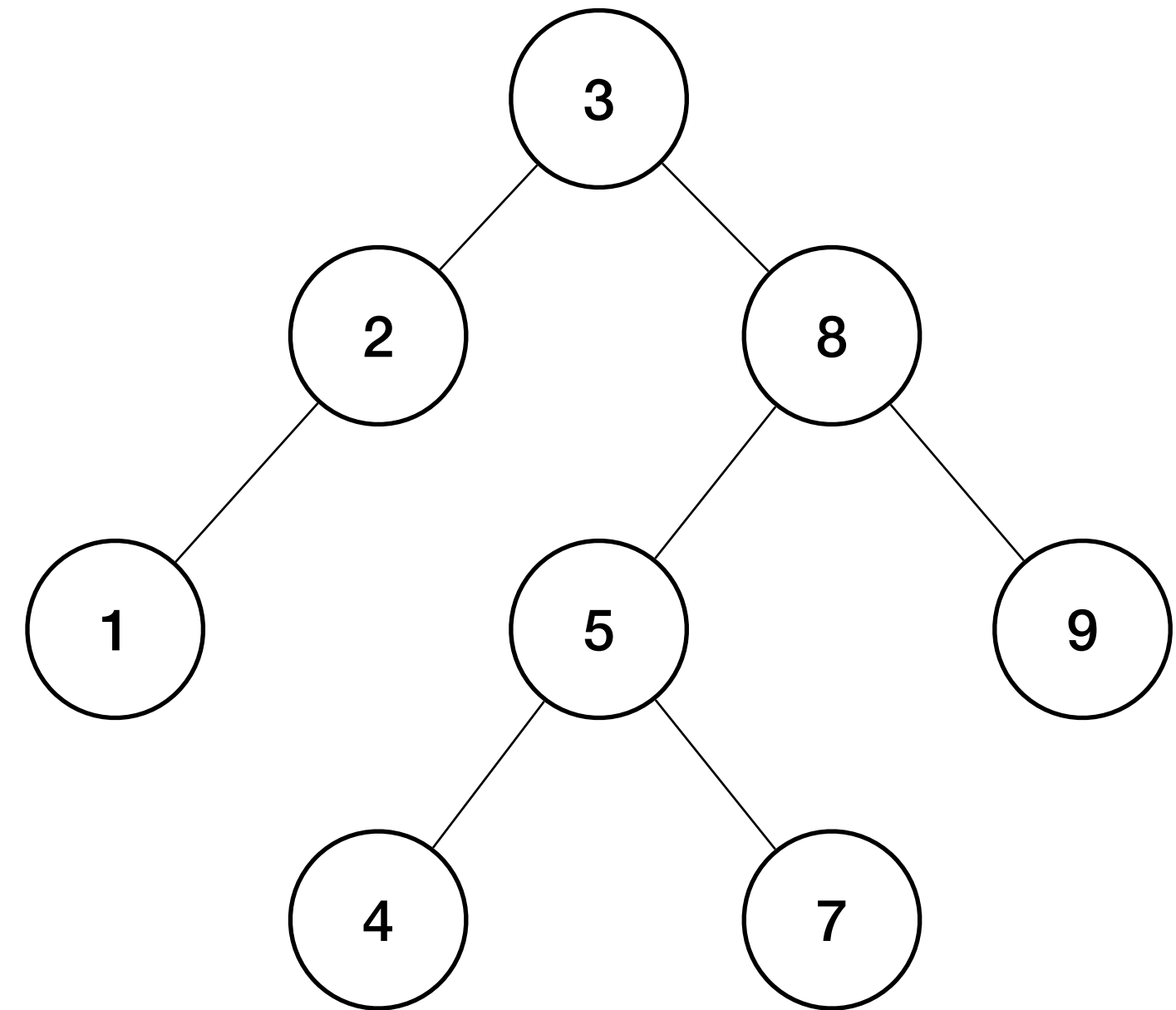
Insertion in a BST

```
node * insert(node *cursor, int data){  
  if (cursor==NULL)  
    return newNode(data);  
  else{  
    if (data < cursor->data)  
      cursor->left = insert(cursor->left, data);  
    else  
      cursor->right =  
      return cursor;  
  }  
}
```

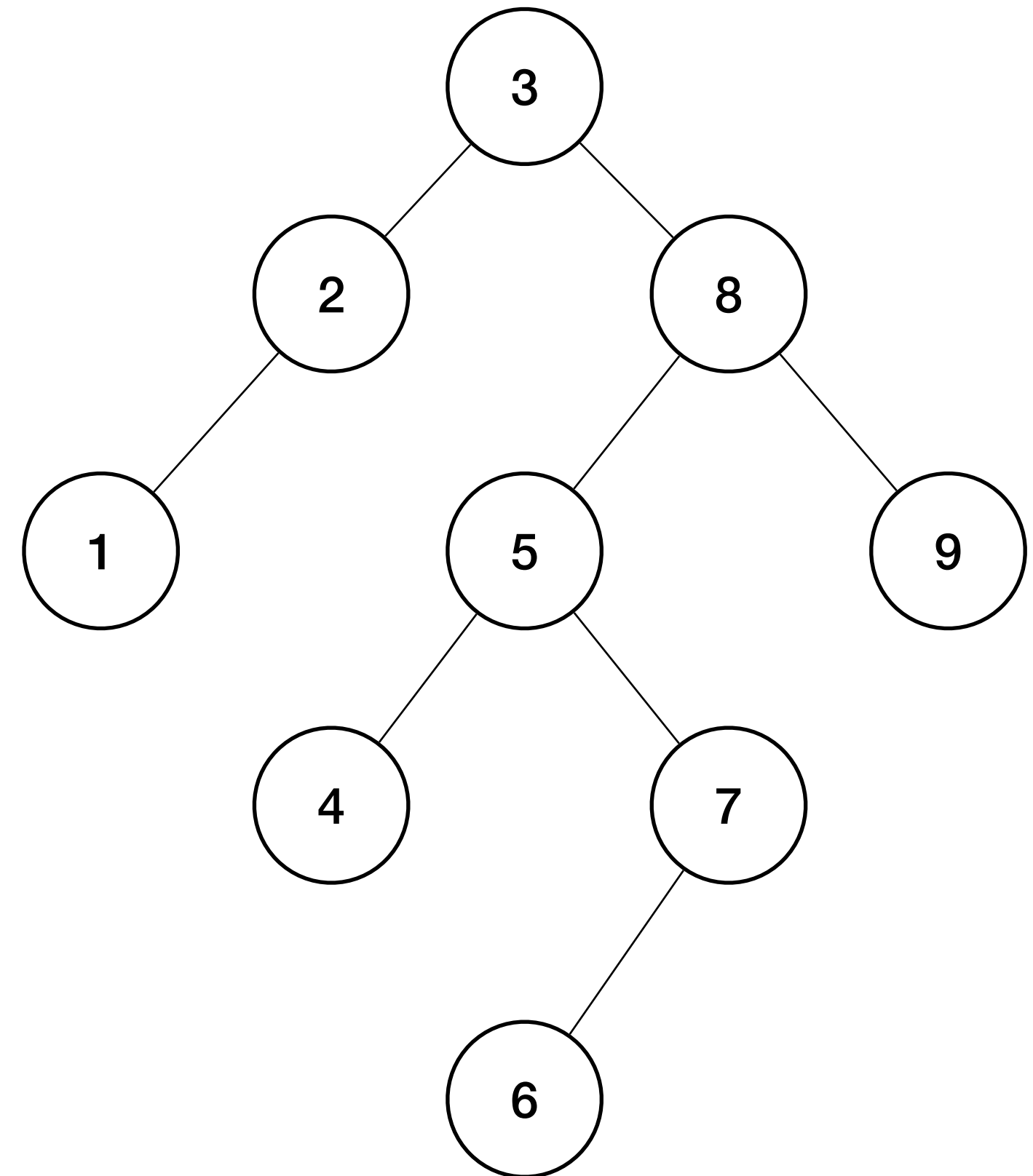


Insertion in a BST

```
node * insert(node *cursor, int data){  
  if (cursor==NULL)  
    return newNode(data);  
  else{  
    if (data < cursor->data)  
      cursor->left = insert(cursor->left, data);  
    else  
      cursor->right = insert(cursor->right, data);  
    return cursor;  
  }  
}
```

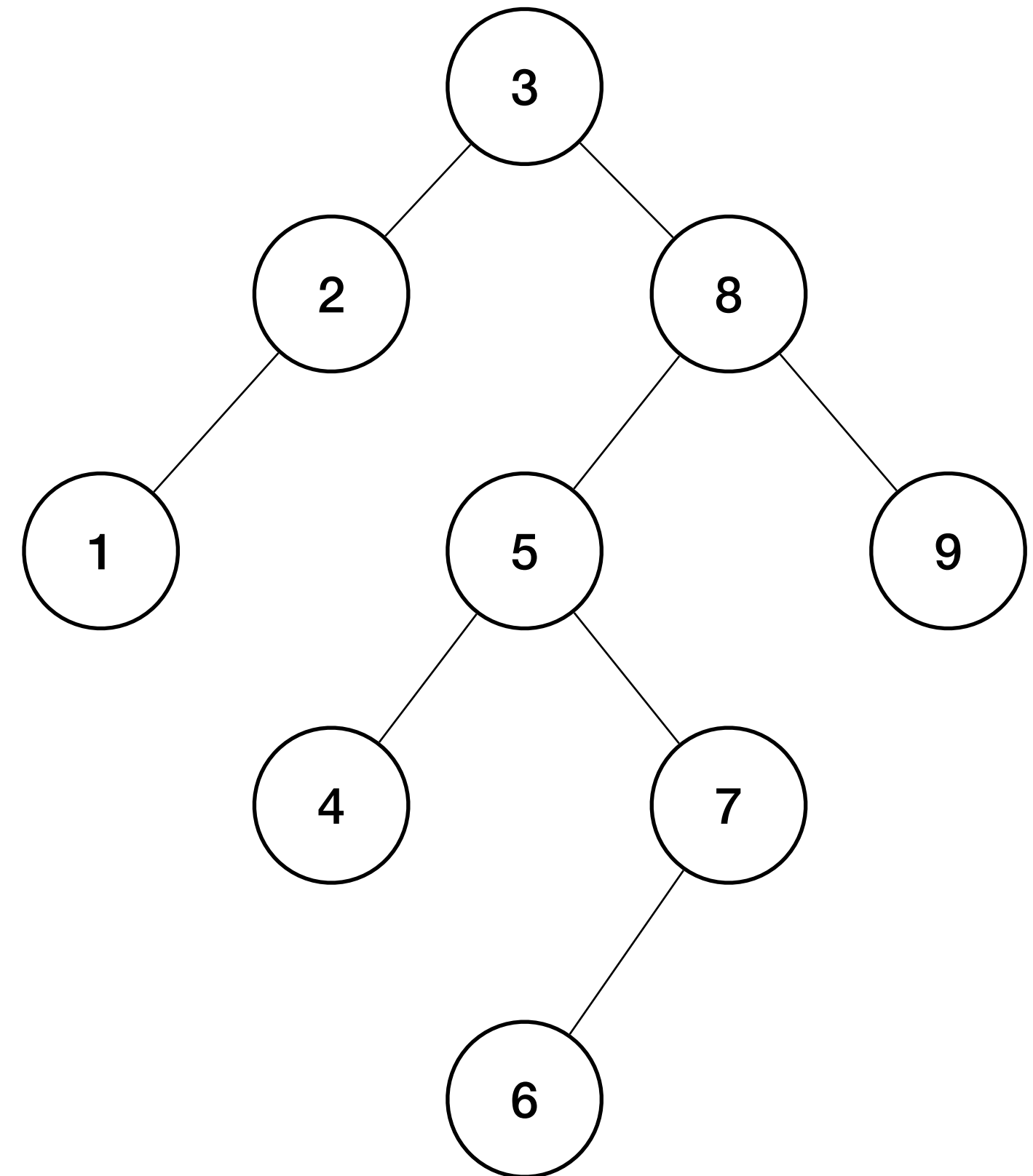


Finding height of a tree



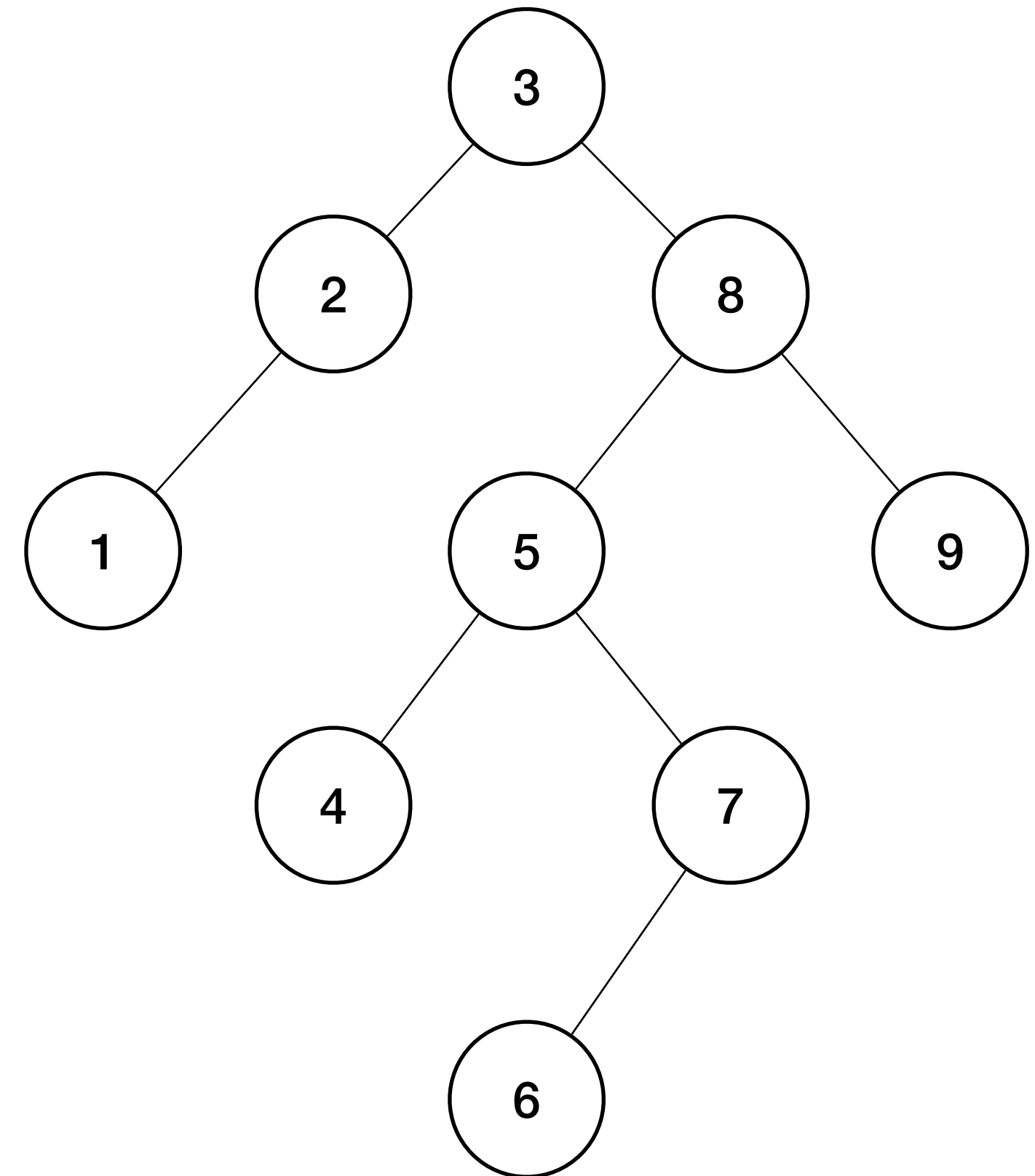
Finding height of a tree

- Height is length of *longest* path from root to leaf(s)



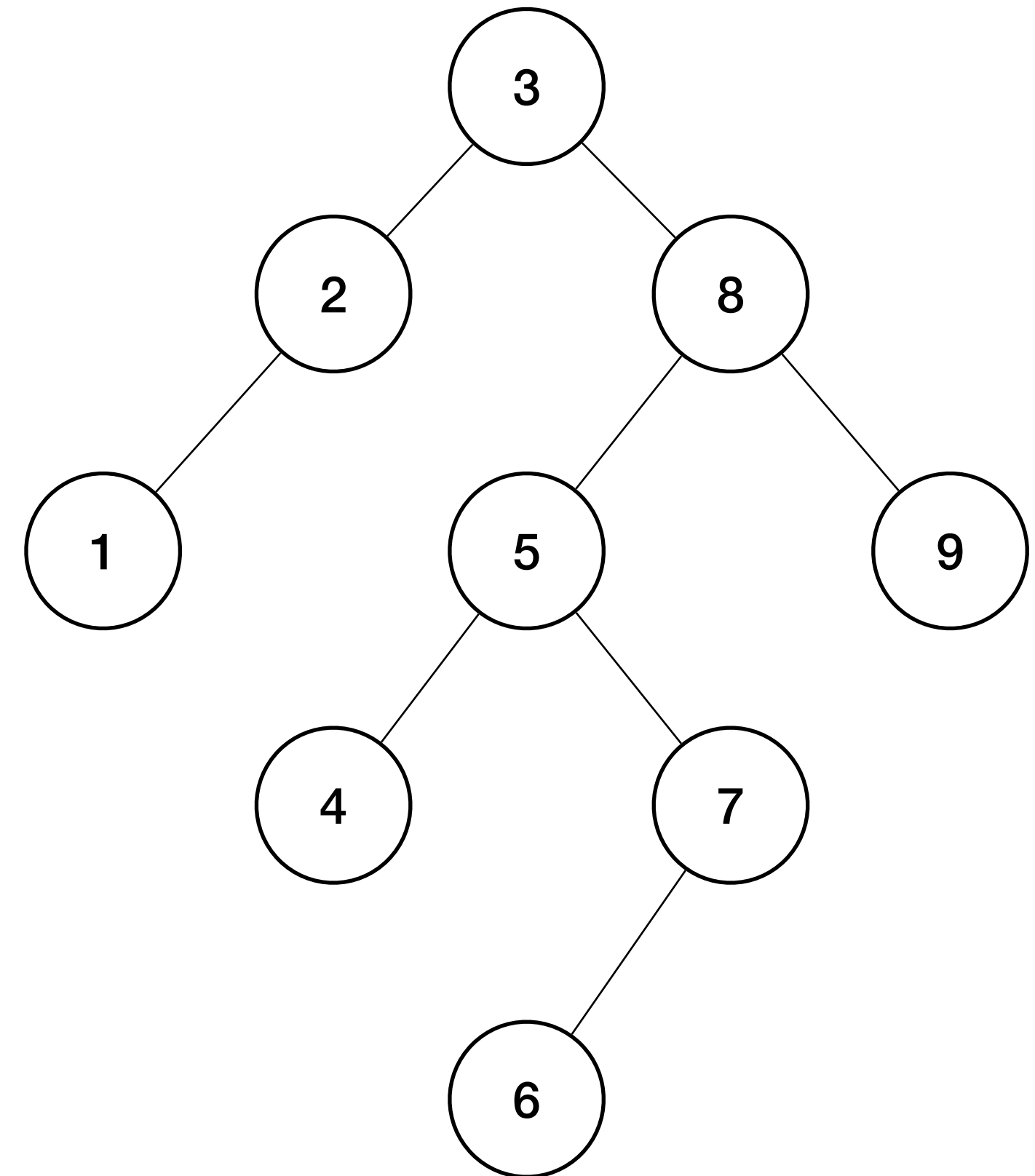
Finding height of a tree

- Height is length of *longest* path from root to leaf(s)
- Recursively calculate: 1 + height of L/R subtree(s)



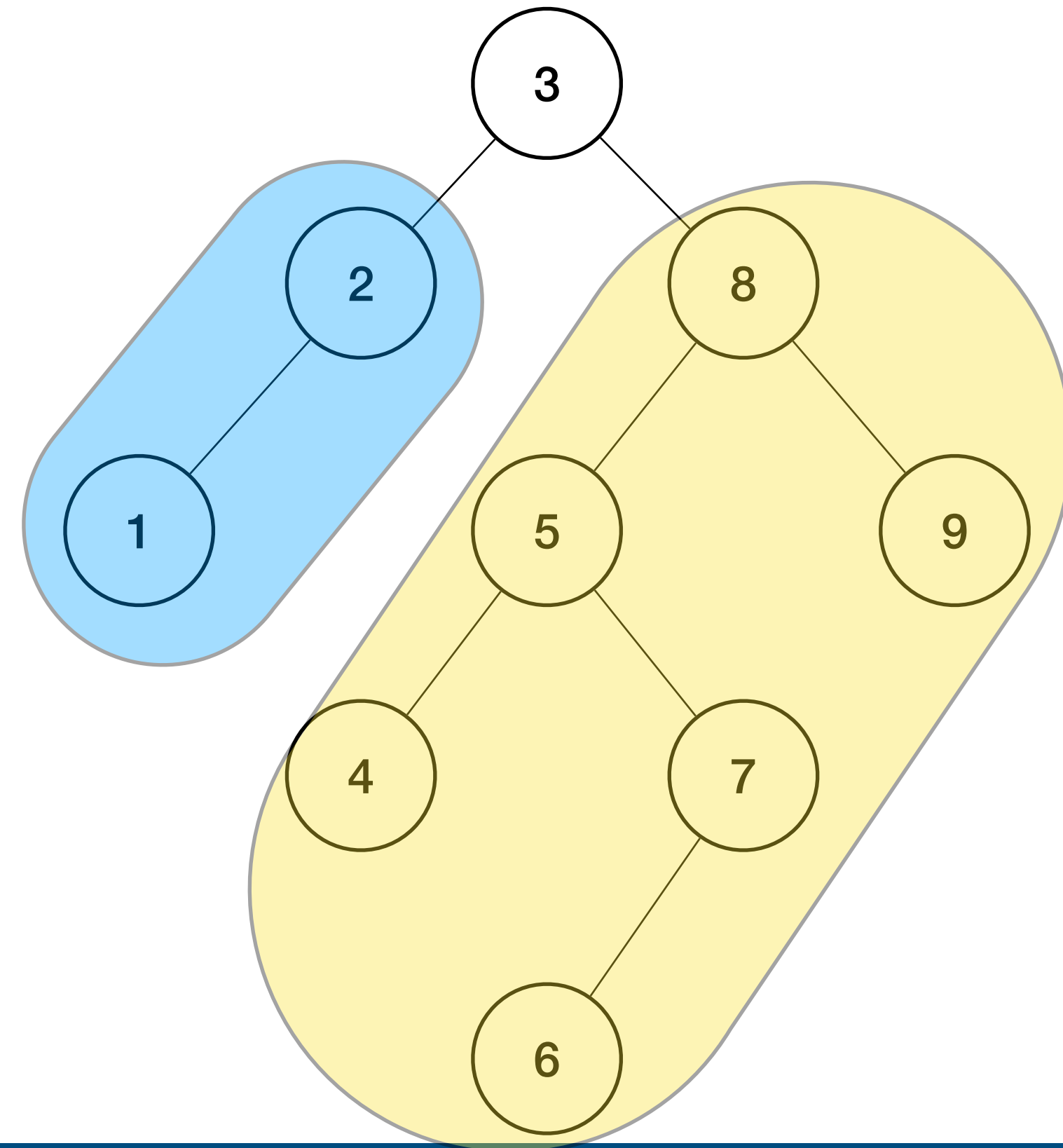
Finding height of a tree

- Height is length of *longest* path from root to leaf(s)
- Recursively calculate: 1 + height of L/R subtree(s)
- Take maximum at each step



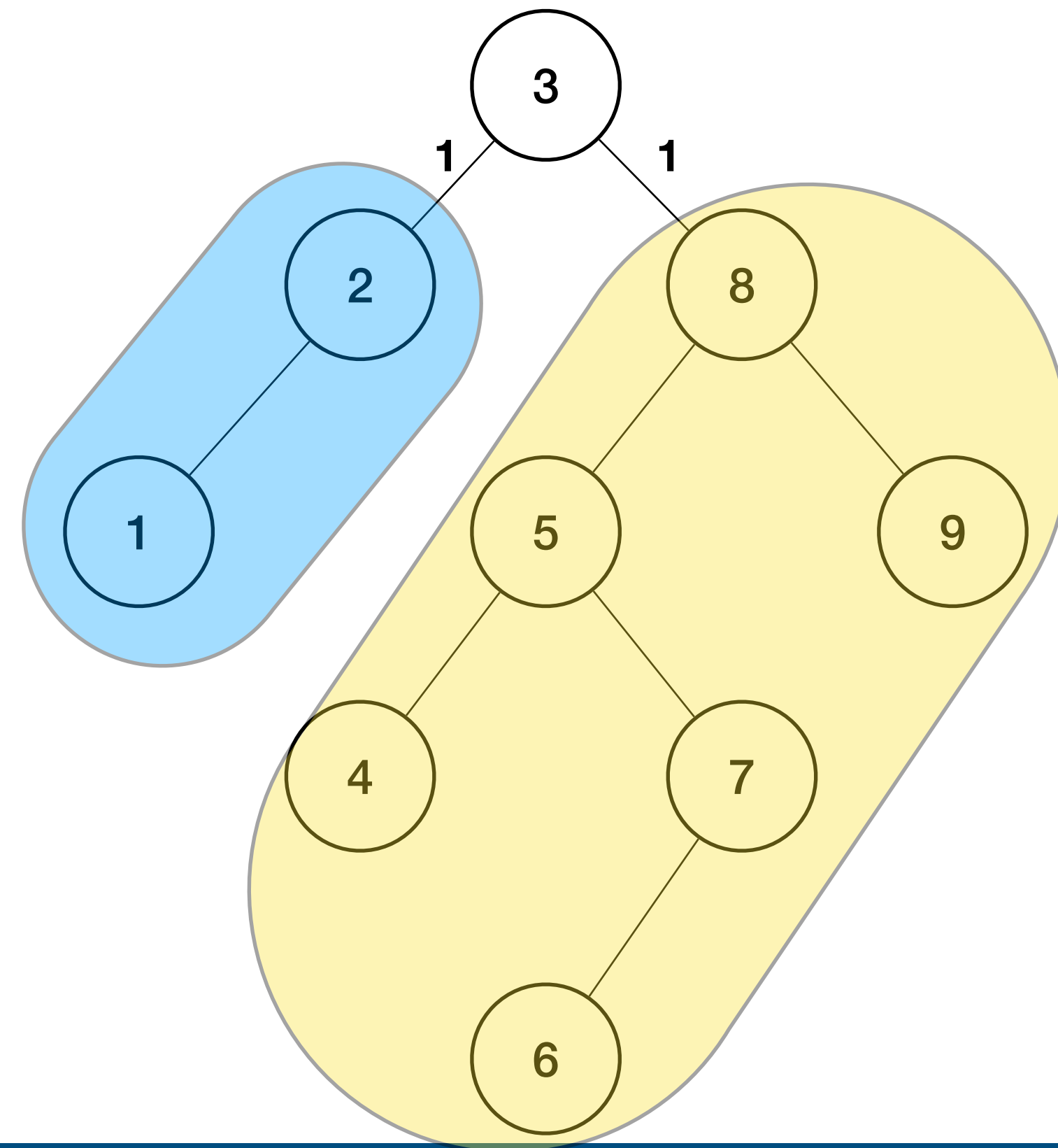
Finding height of a tree

- Height is length of *longest* path from root to leaf(s)
- Recursively calculate: $1 +$ height of L/R subtree(s)
- Take maximum at each step



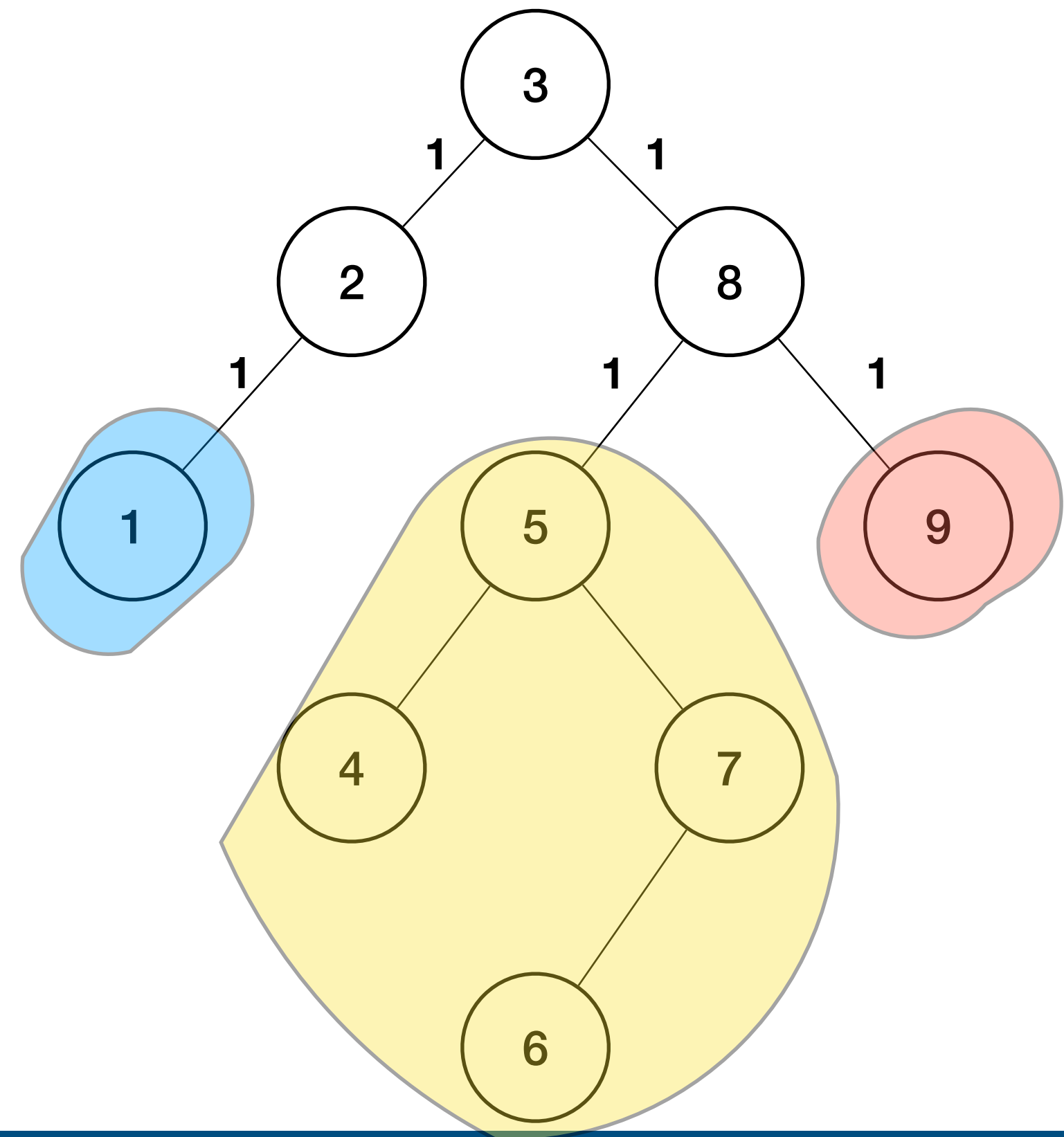
Finding height of a tree

- Height is length of *longest* path from root to leaf(s)
- Recursively calculate: $1 +$ height of L/R subtree(s)
- Take maximum at each step



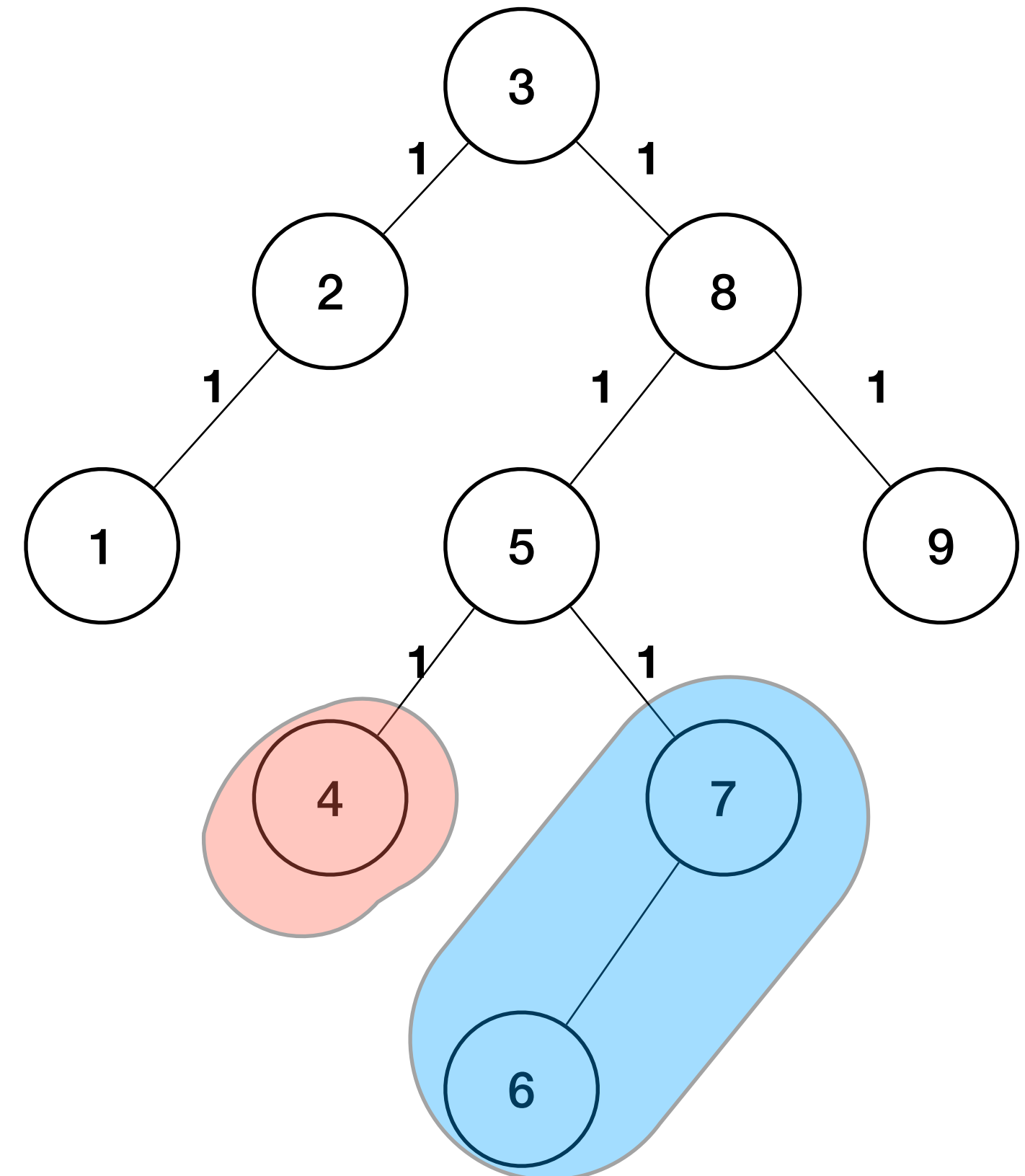
Finding height of a tree

- Height is length of *longest* path from root to leaf(s)
- Recursively calculate: $1 +$ height of L/R subtree(s)
- Take maximum at each step



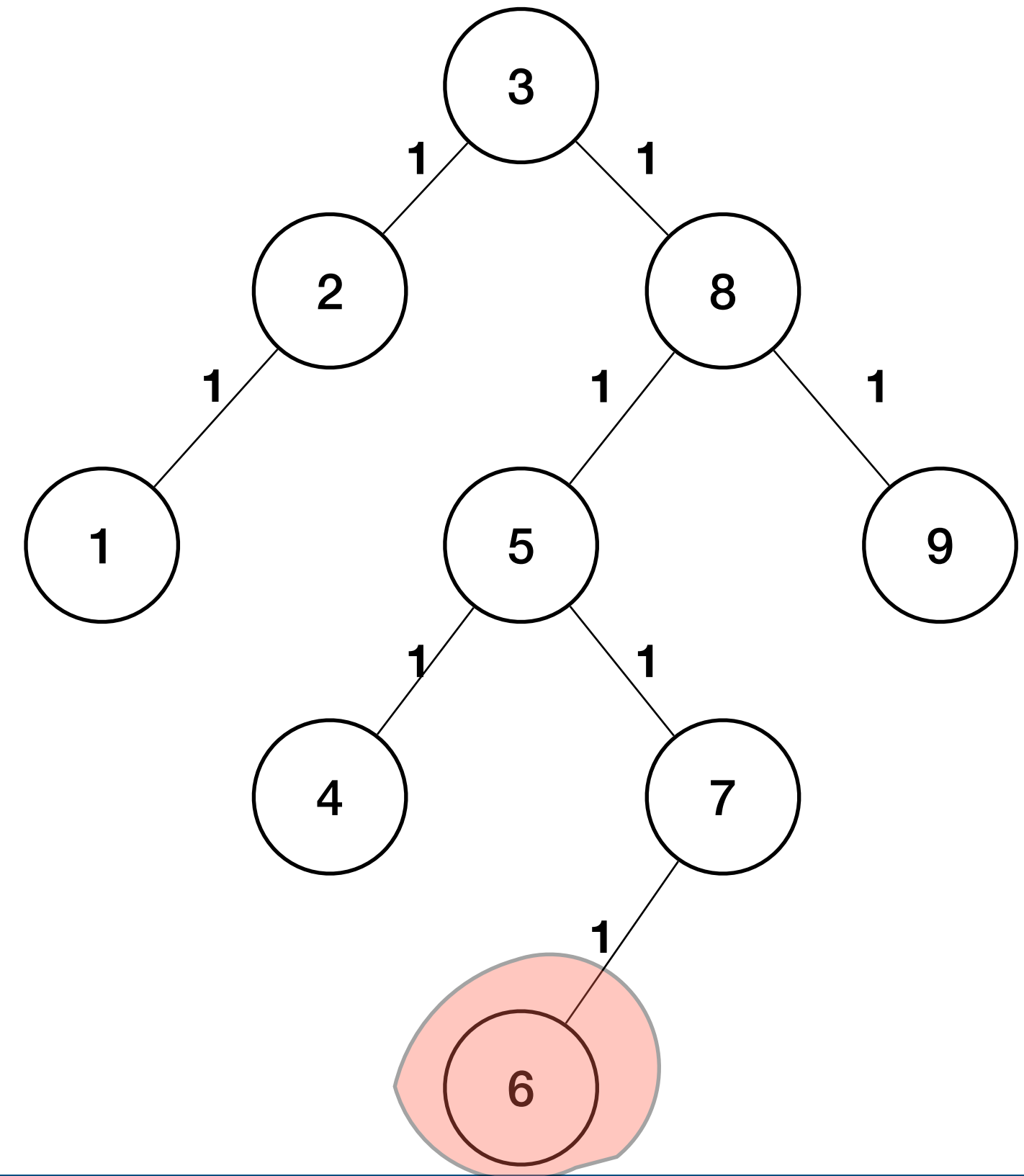
Finding height of a tree

- Height is length of *longest* path from root to leaf(s)
- Recursively calculate: 1 + height of L/R subtree(s)
- Take maximum at each step



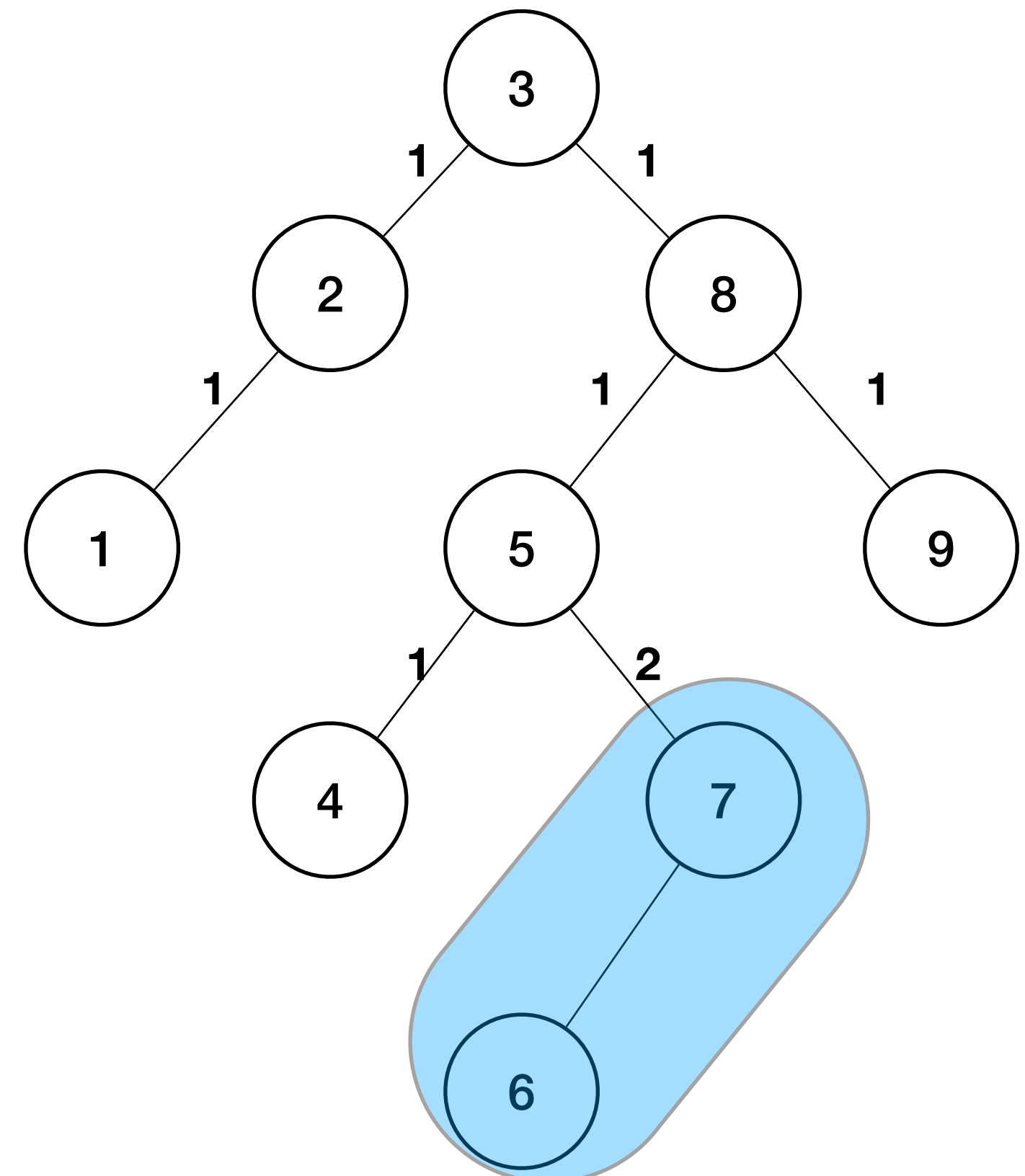
Finding height of a tree

- Height is length of *longest* path from root to leaf(s)
- Recursively calculate: $1 +$ height of L/R subtree(s)
- Take maximum at each step



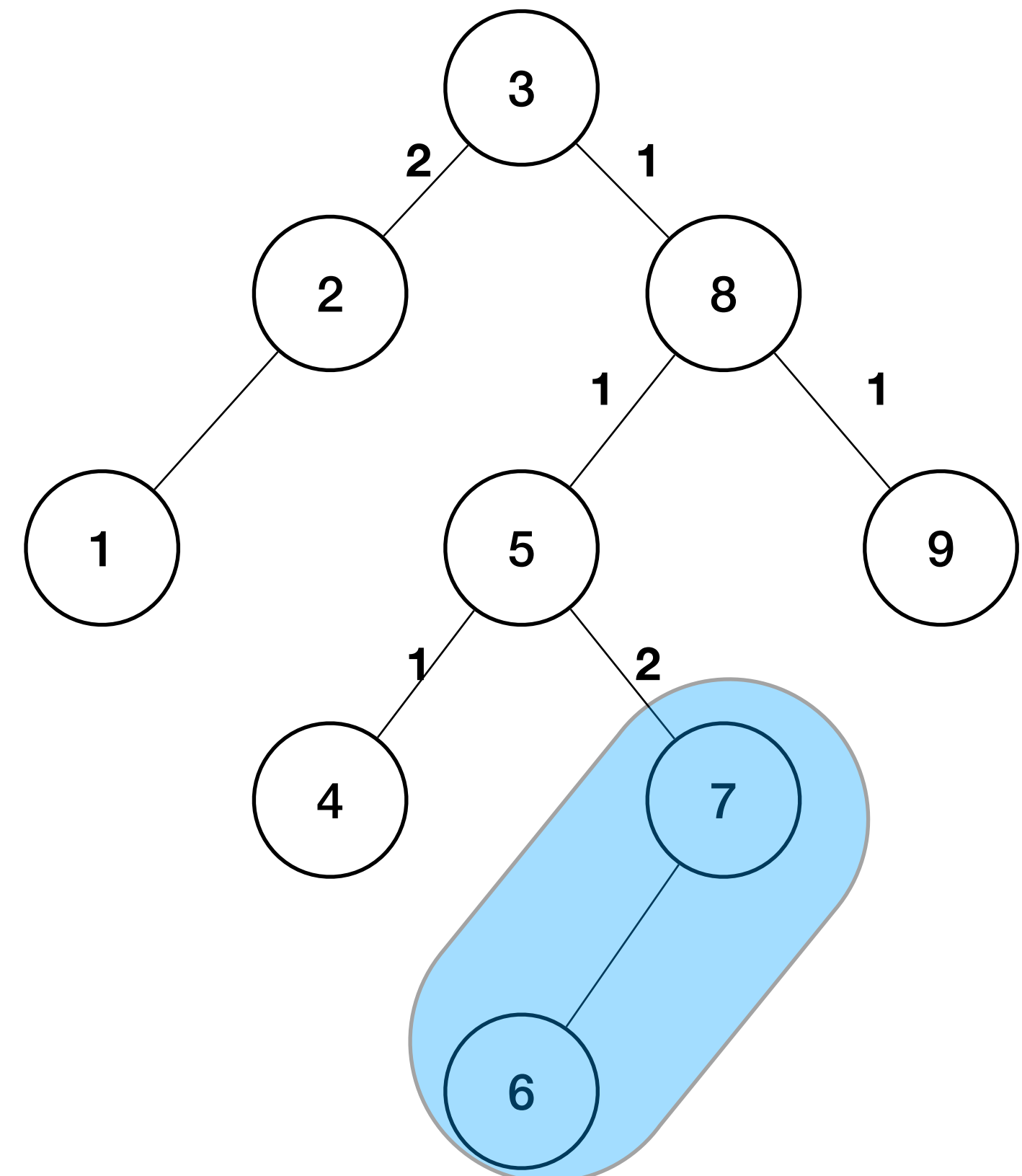
Finding height of a tree

- Height is length of *longest* path from root to leaf(s)
- Recursively calculate: 1 + height of L/R subtree(s)
- Take maximum at each step



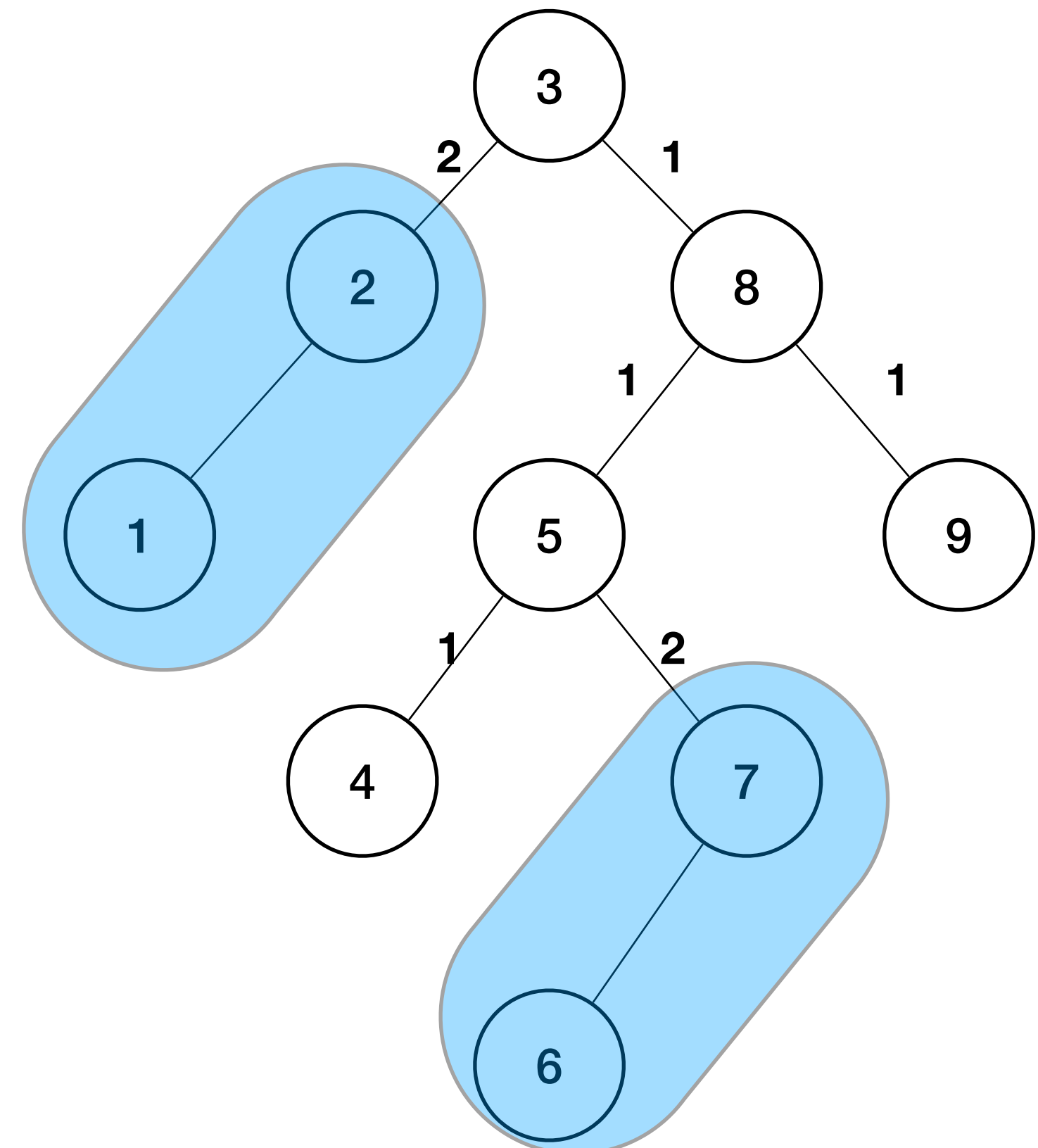
Finding height of a tree

- Height is length of *longest* path from root to leaf(s)
- Recursively calculate: 1 + height of L/R subtree(s)
- Take maximum at each step



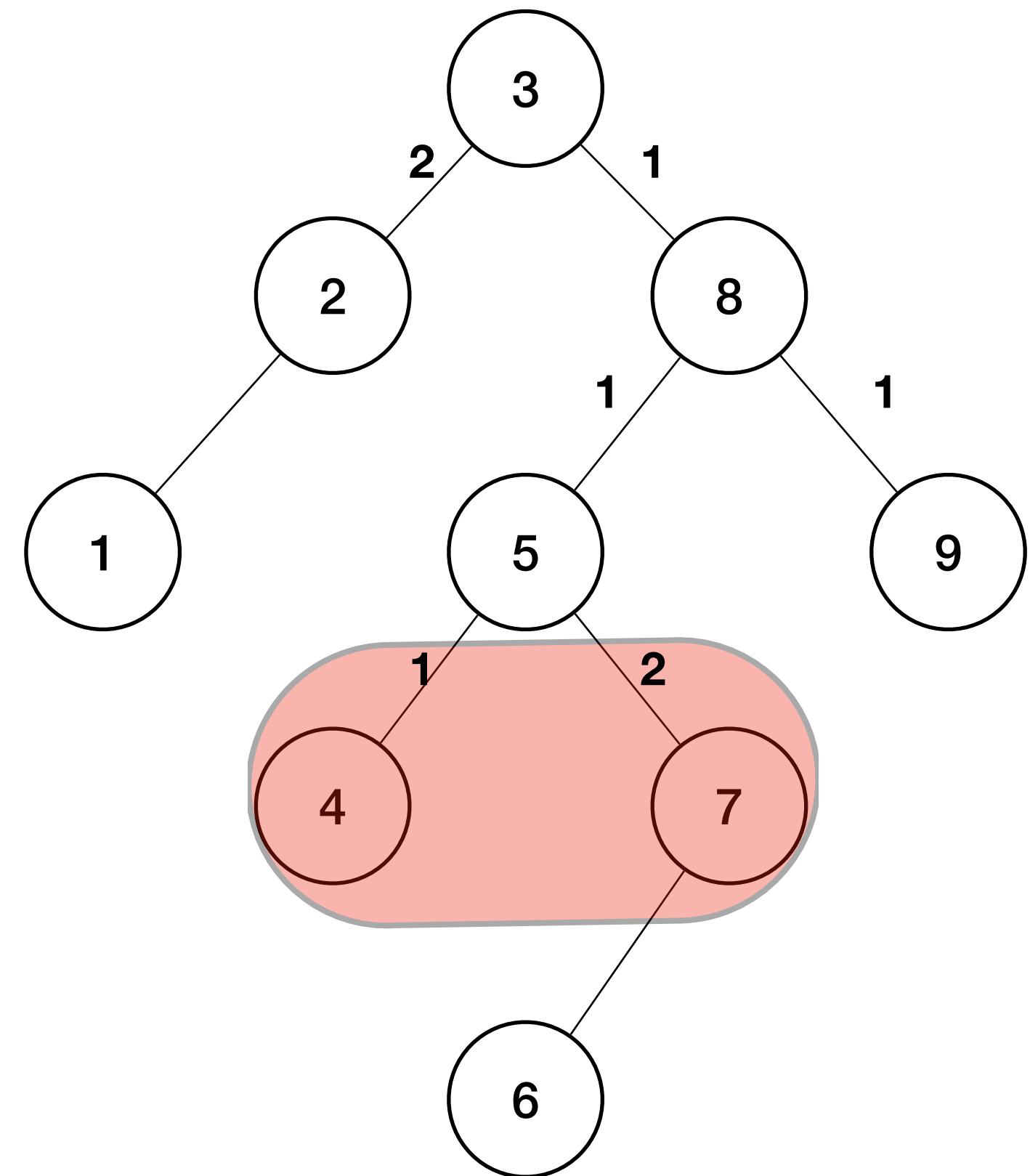
Finding height of a tree

- Height is length of *longest* path from root to leaf(s)
- Recursively calculate: 1 + height of L/R subtree(s)
- Take maximum at each step



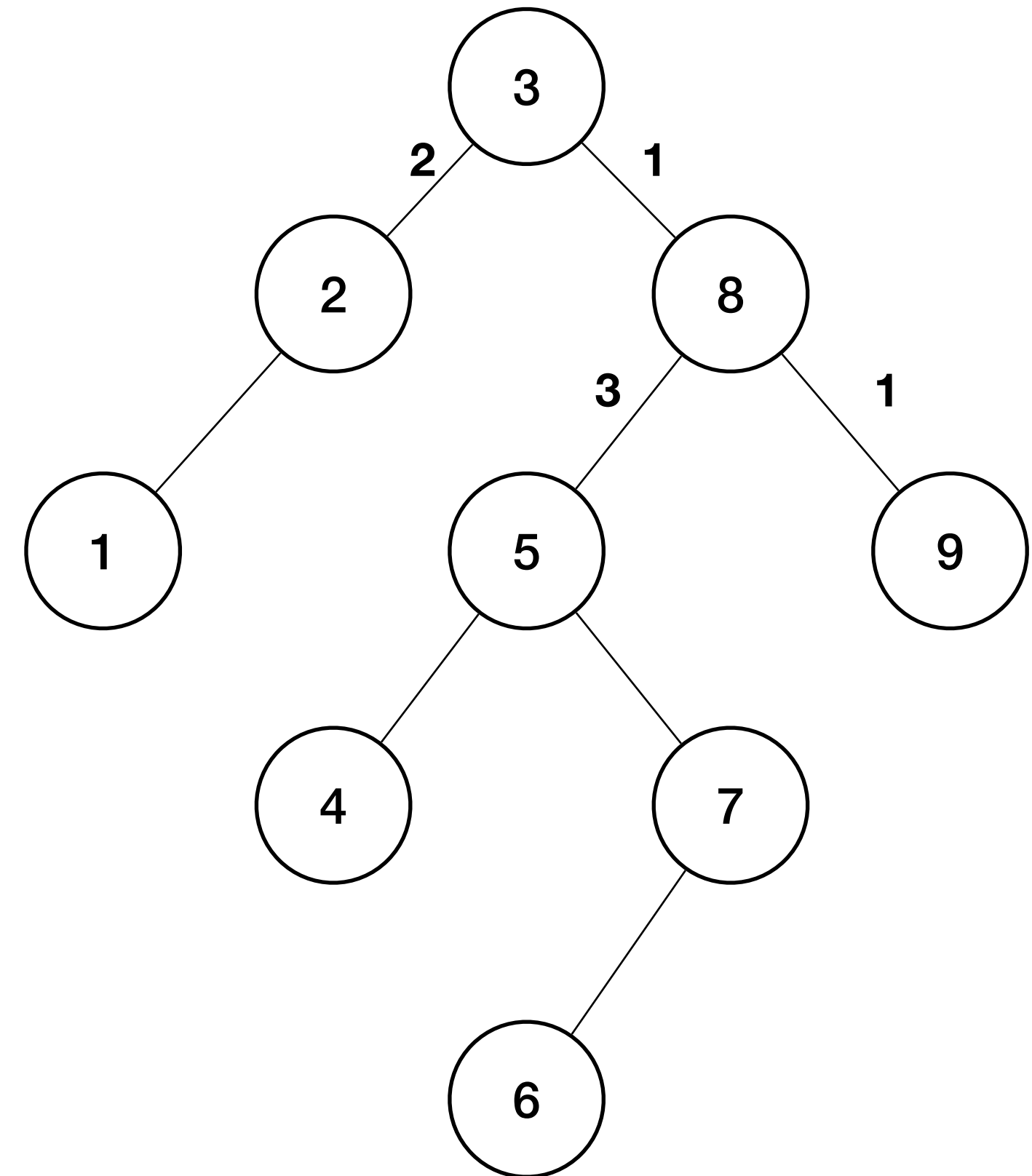
Finding height of a tree

- Height is length of *longest* path from root to leaf(s)
- Recursively calculate: 1 + height of L/R subtree(s)
- Take maximum at each step



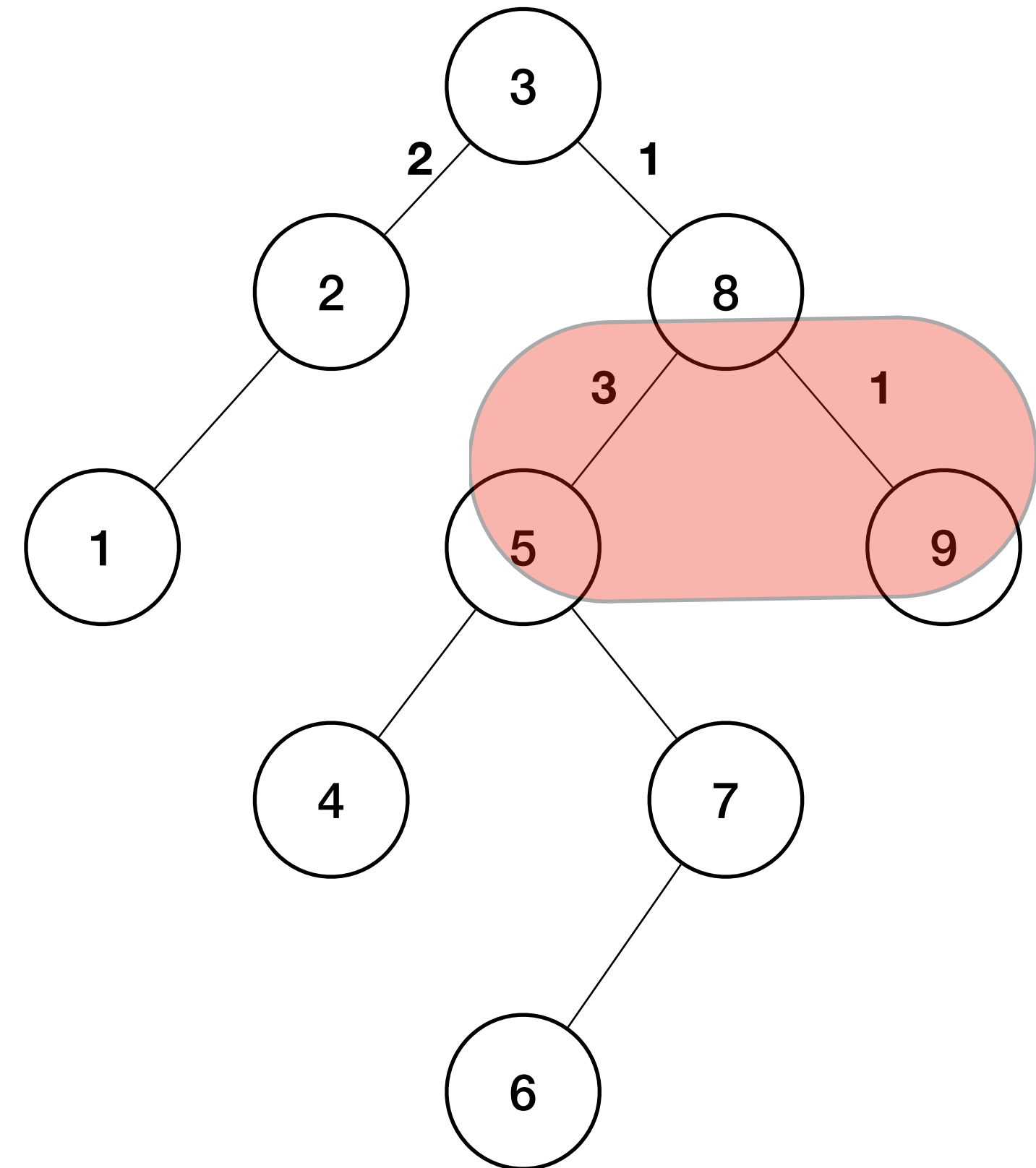
Finding height of a tree

- Height is length of *longest* path from root to leaf(s)
- Recursively calculate: $1 +$ height of L/R subtree(s)
- Take maximum at each step



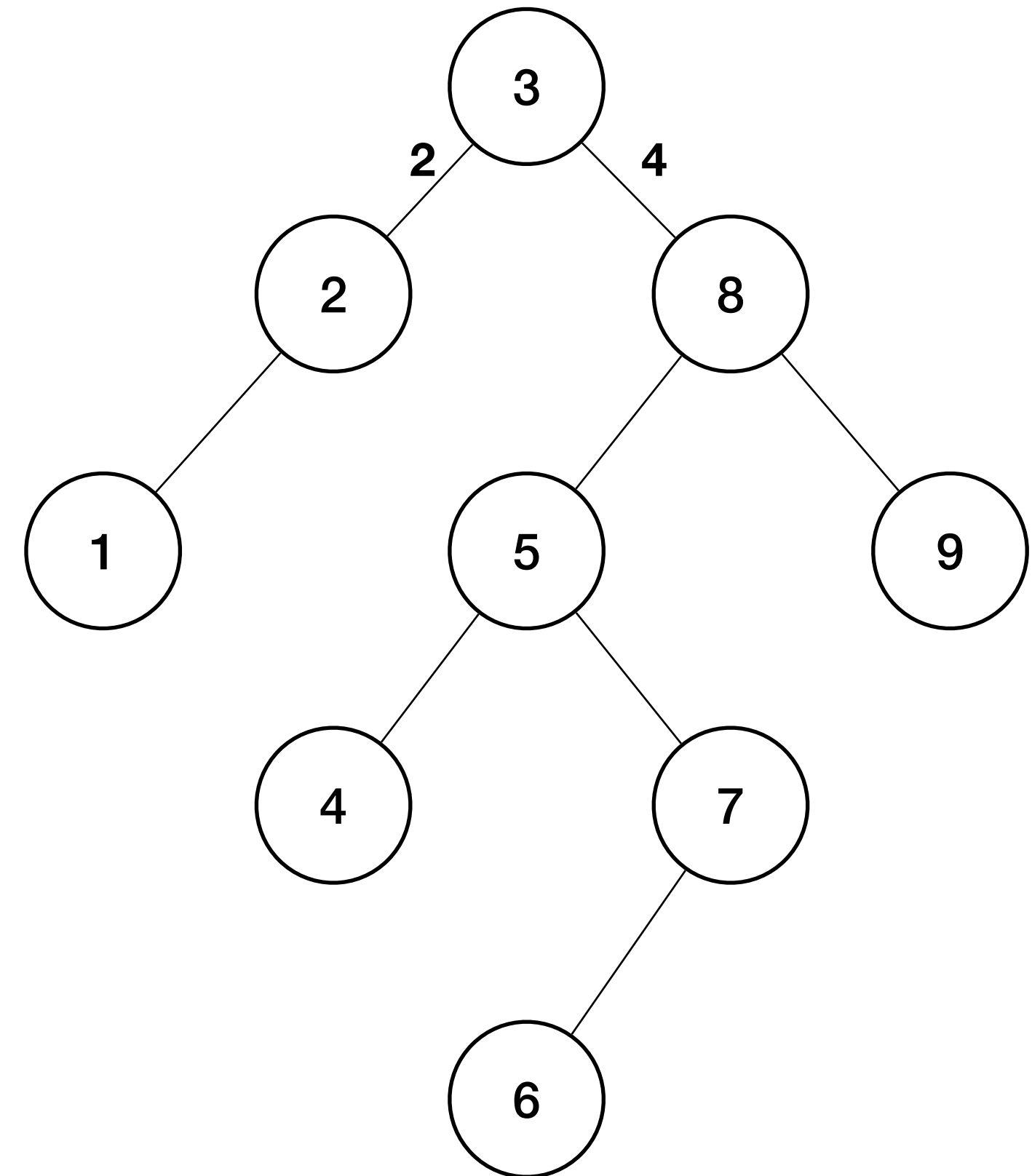
Finding height of a tree

- Height is length of *longest* path from root to leaf(s)
- Recursively calculate: $1 +$ height of L/R subtree(s)
- Take maximum at each step



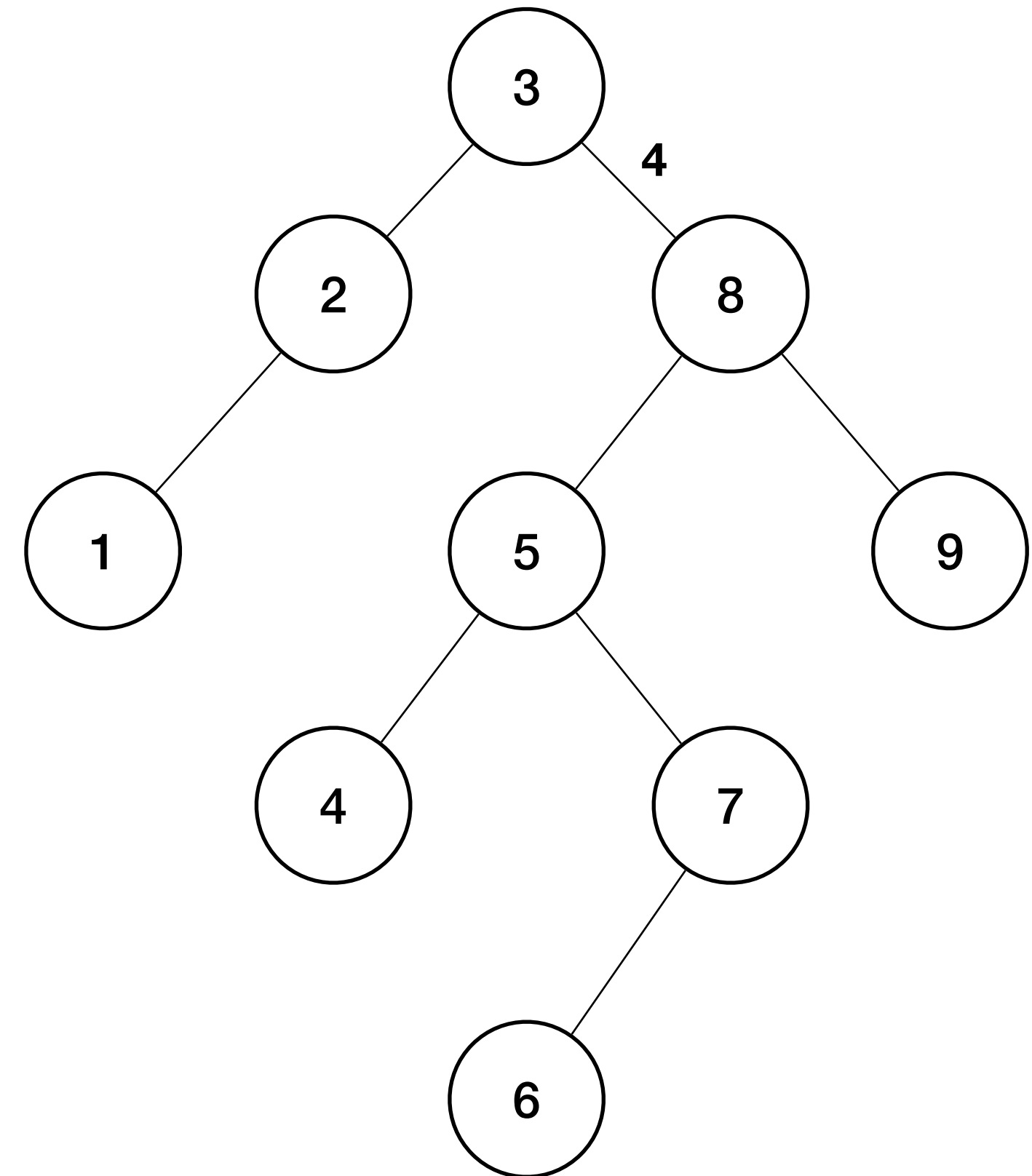
Finding height of a tree

- Height is length of *longest* path from root to leaf(s)
- Recursively calculate: 1 + height of L/R subtree(s)
- Take maximum at each step



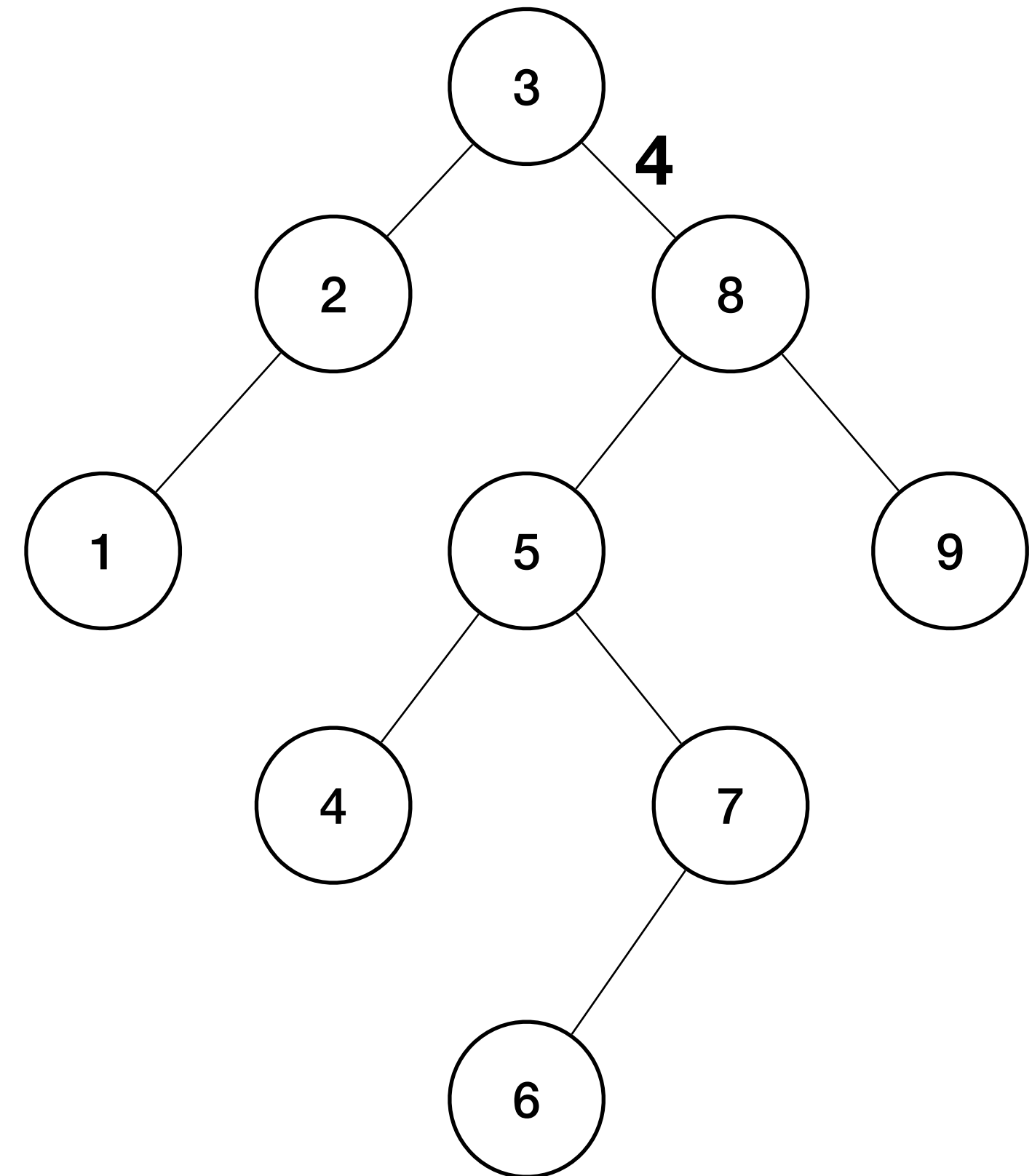
Finding height of a tree

- Height is length of *longest* path from root to leaf(s)
- Recursively calculate: 1 + height of L/R subtree(s)
- Take maximum at each step



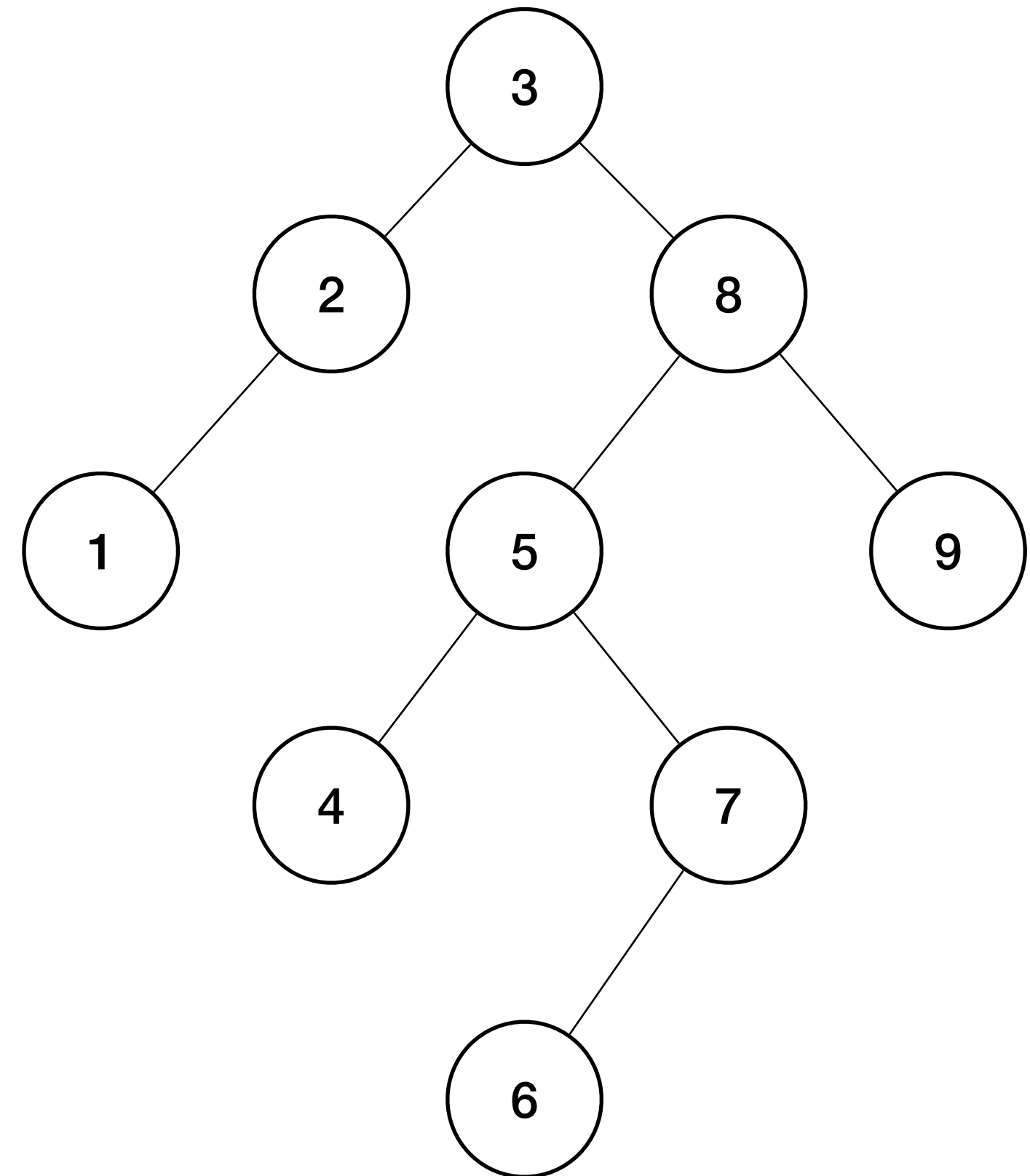
Finding height of a tree

- Height is length of *longest* path from root to leaf(s)
- Recursively calculate: 1 + height of L/R subtree(s)
- Take maximum at each step



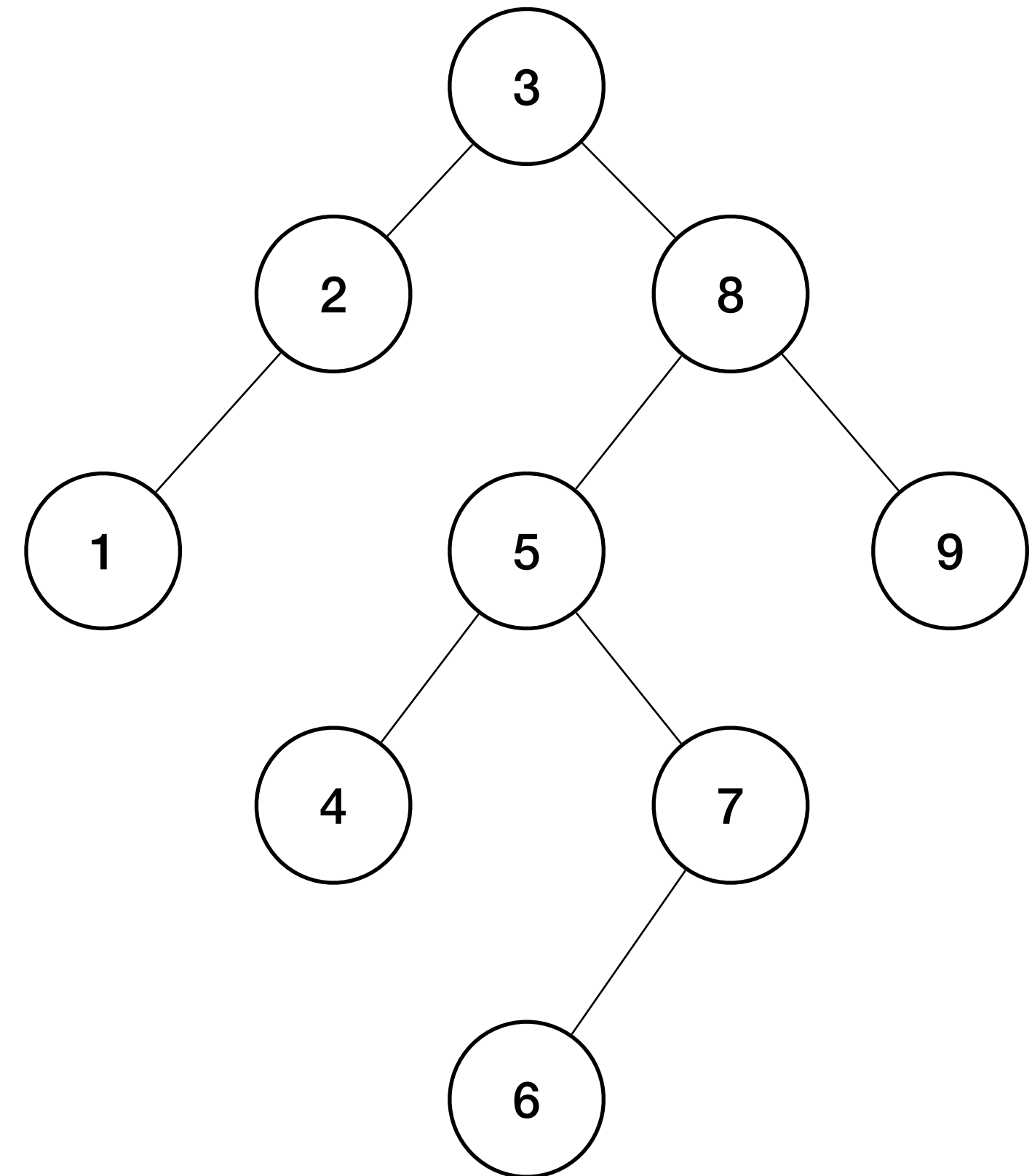
Find height of a tree

```
int tree_height(node *cursor){  
    int lh, rh;  
    if (cursor==NULL)  
  
    else{  
        lh =  
        rh =  
        return  
    }  
}
```



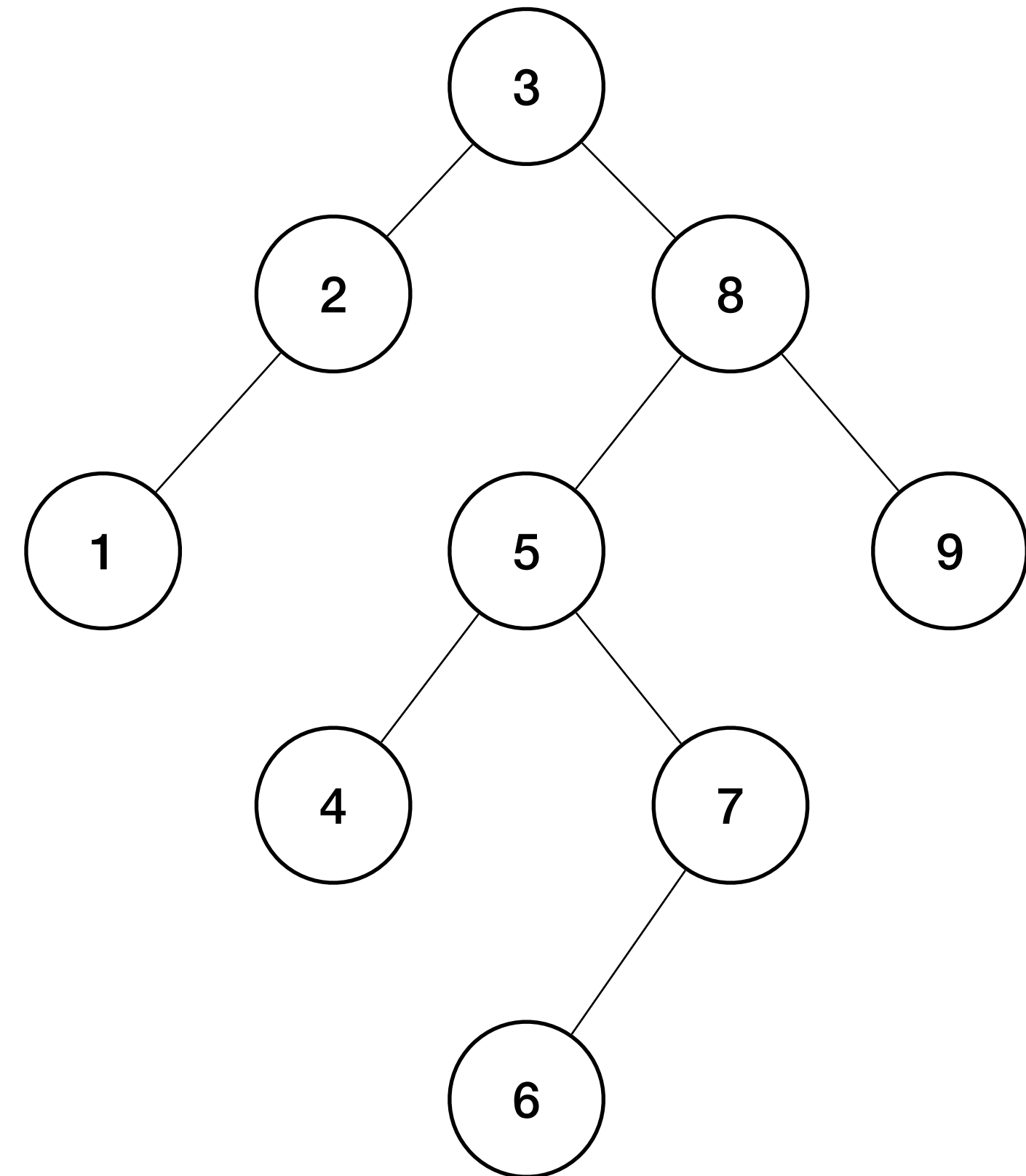
Find height of a tree

```
int tree_height(node *cursor){  
    int lh, rh;  
    if (cursor==NULL)  
        return -1;  
    else{  
        lh =  
        rh =  
        return  
    }  
}
```

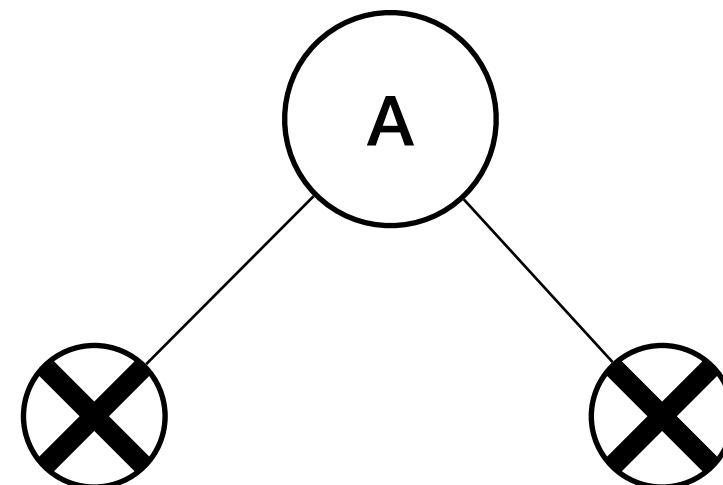


Find height of a tree

```
int tree_height(node *cursor){  
    int lh, rh;  
    if (cursor==NULL)  
        return -1;  
    else{  
        lh =  
        rh =  
        return  
    }  
}
```

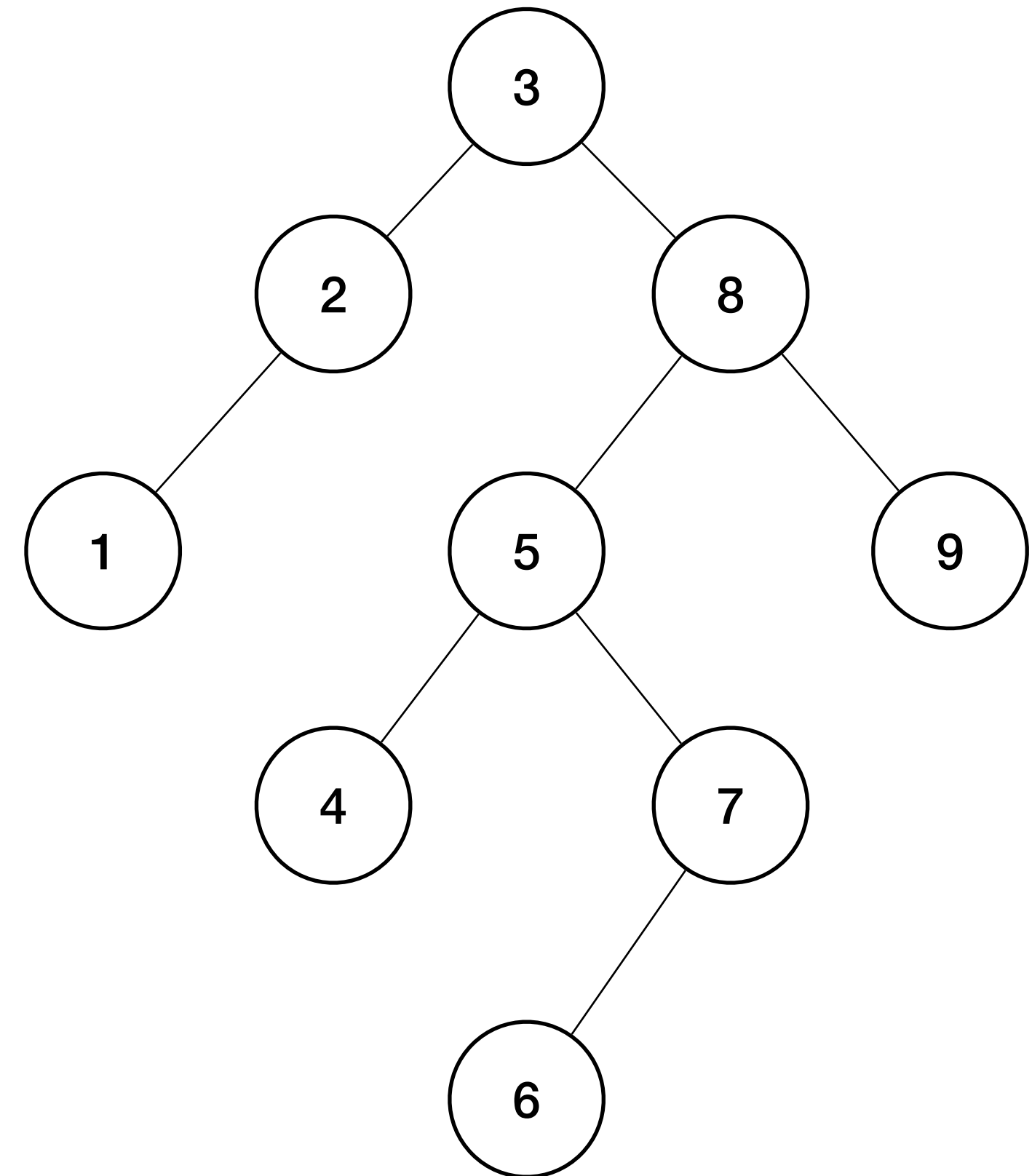


What should be height of single node?

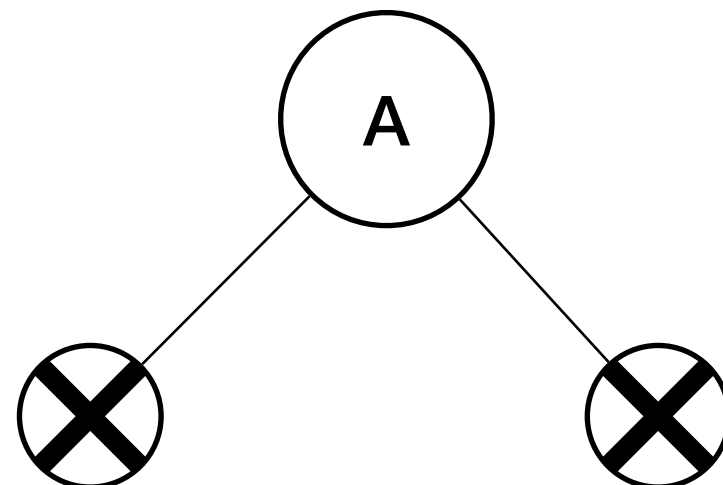


Find height of a tree

```
int tree_height(node *cursor){  
    int lh, rh;  
    if (cursor==NULL)  
        return -1;  
    else{  
        lh = 1 + tree_height(cursor->left);  
        rh =  
        return  
    }  
}
```

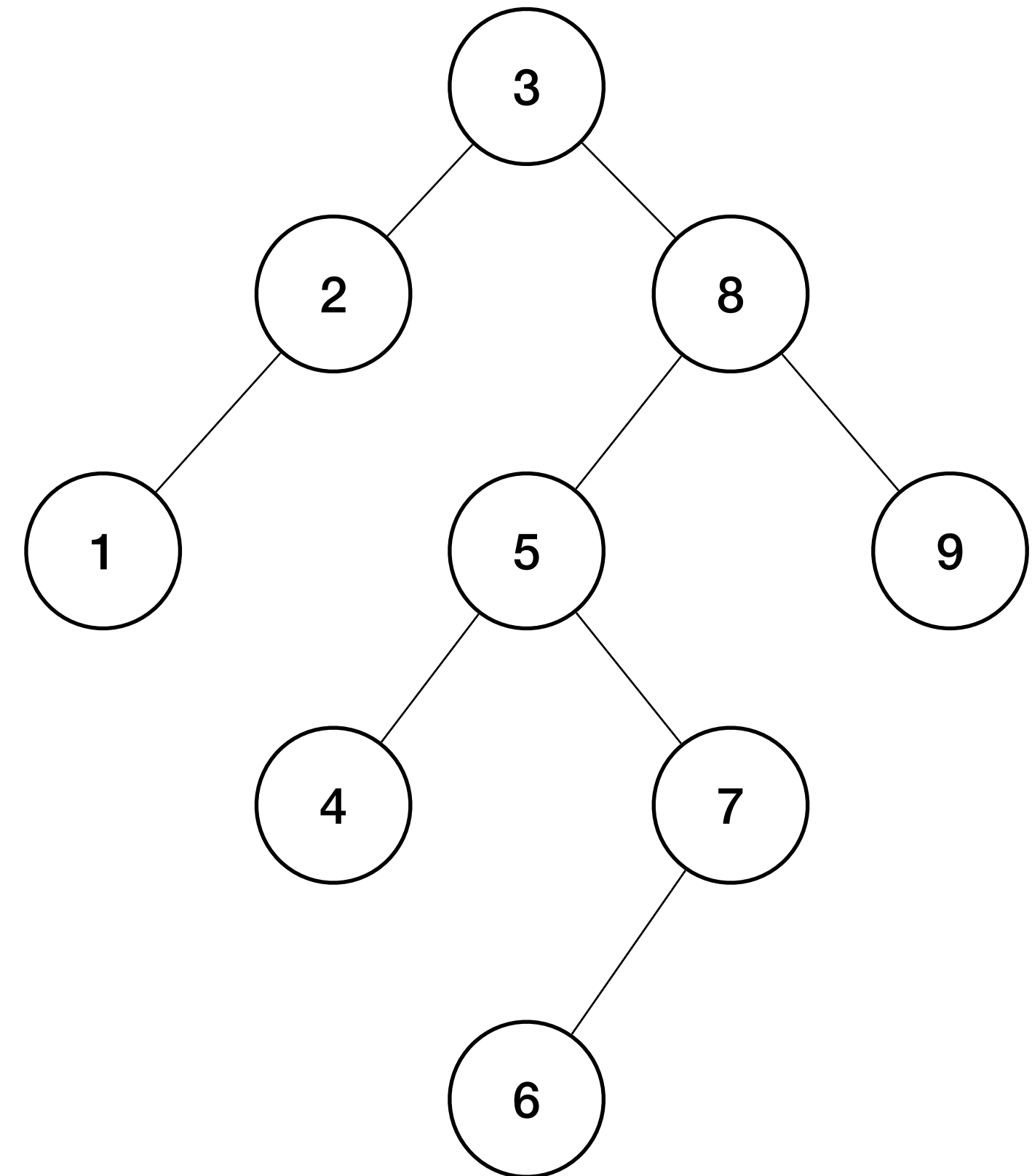


What should be height of single node?

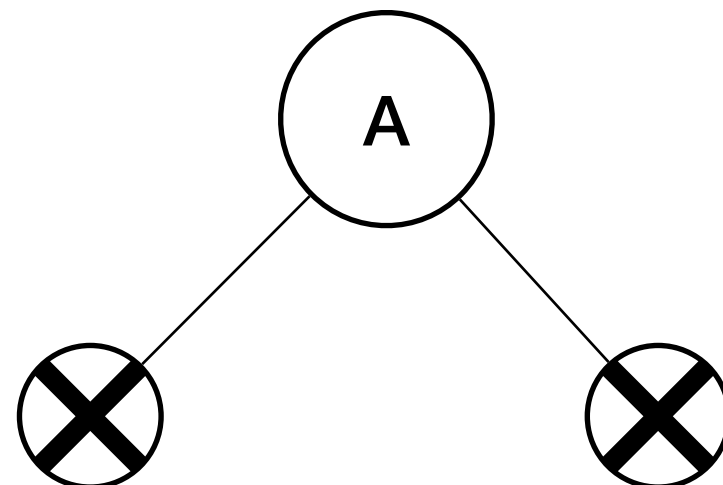


Find height of a tree

```
int tree_height(node *cursor){
    int lh, rh;
    if (cursor==NULL)
        return -1;
    else{
        lh = 1 + tree_height(cursor->left);
        rh = 1 + tree_height(cursor->right);
        return
    }
}
```

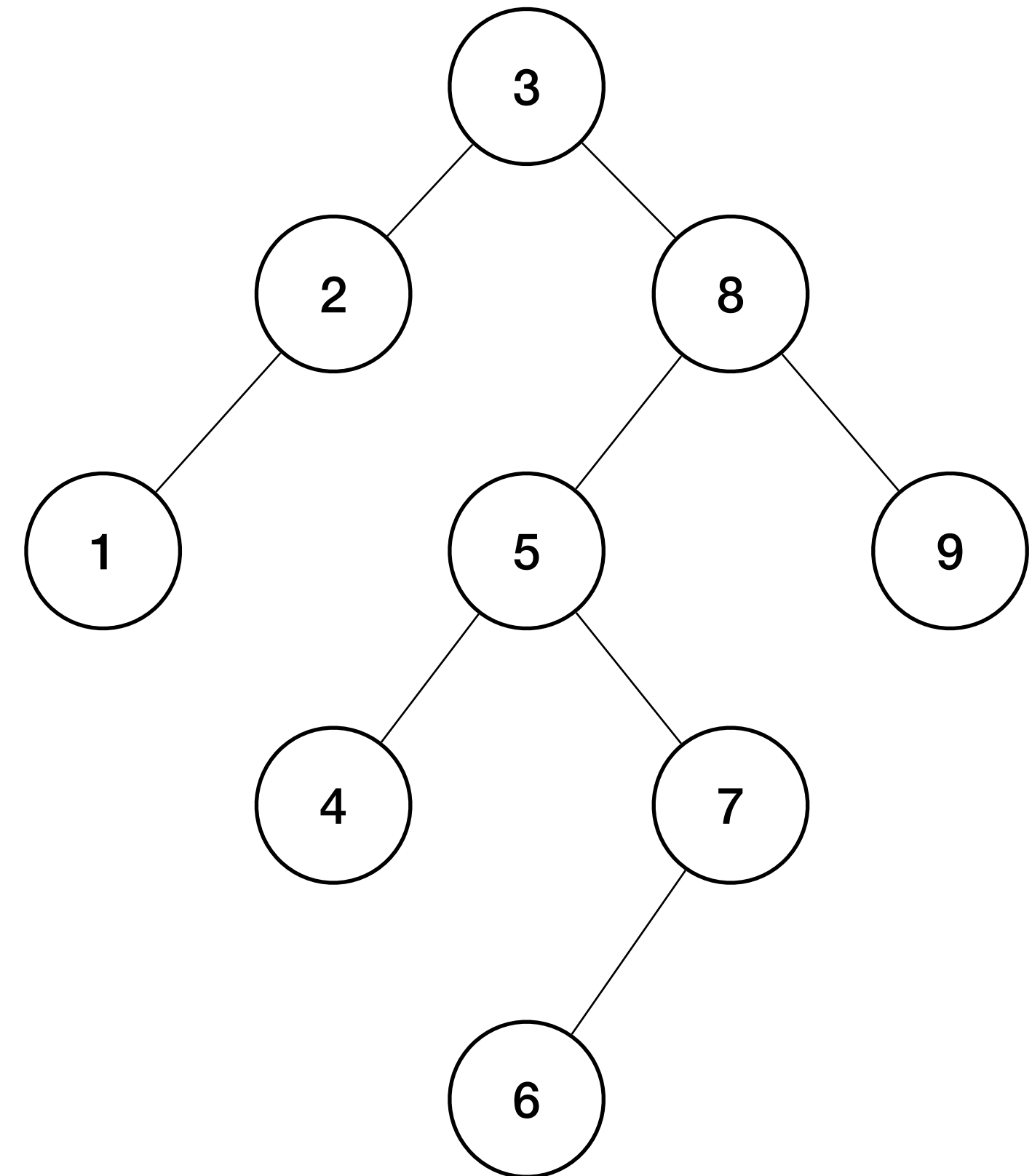


What should be height of single node?

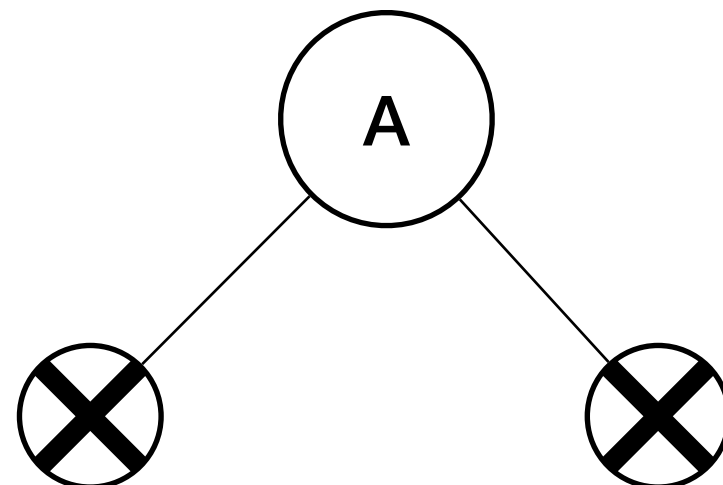


Find height of a tree

```
int tree_height(node *cursor){
    int lh, rh;
    if (cursor==NULL)
        return -1;
    else{
        lh = 1 + tree_height(cursor->left);
        rh = 1 + tree_height(cursor->right);
        return (lh > rh ? lh : rh);
    }
}
```



What should be height of single node?



```
template <typename T>
struct treeNode{
    T data;
    treeNode *left;
    treeNode *right;
};
```

C++ Example

- Using classes in C++, create a BST class and perform or find:

```
template <class N>
class bst{
private:
    ...
    ...
    ...
public:
    bst();
```

```
template <typename T>
struct treeNode{
    T data;
    treeNode *left;
    treeNode *right;
};
```

C++ Example

- Using classes in C++, create a BST class and perform or find:
 - Insertion

```
template <class N>
class bst{
private:
    ...
    ...
    ...
public:
    bst();
    void insert(N data);
```

```
template <typename T>
struct treeNode{
    T data;
    treeNode *left;
    treeNode *right;
};
```

C++ Example

- Using classes in C++, create a BST class and perform or find:
 - Insertion
 - Searching

```
template <class N>
class bst{
private:
    ...
    ...
    ...
public:
    bst();
    void insert(N data);
    treeNode<N> *search(N data);
```

```
template <typename T>
struct treeNode{
    T data;
    treeNode *left;
    treeNode *right;
};
```

C++ Example

- Using classes in C++, create a BST class and perform or find:
 - Insertion
 - Searching
 - Traversal

```
template <class N>
class bst{
private:
    ...
    ...
    ...
public:
    bst();
    void insert(N data);
    treeNode<N> *search(N data);
    void inorder();
```

```
template <typename T>
struct treeNode{
    T data;
    treeNode *left;
    treeNode *right;
};
```

C++ Example

- Using classes in C++, create a BST class and perform or find:
 - Insertion
 - Searching
 - Traversal
 - Vectorization

```
template <class N>
class bst{
private:
    ...
    ...
    ...
public:
    bst();
    void insert(N data);
    treeNode<N> *search(N data);
    void inorder();
    vector<N> vectorize();
```

```
template <typename T>
struct treeNode{
    T data;
    treeNode *left;
    treeNode *right;
};
```

C++ Example

- Using classes in C++, create a BST class and perform or find:
 - Insertion
 - Searching
 - Traversal
 - Vectorization
 - Size of tree (# of nodes)

```
template <class N>
class bst{
private:
    ...
    ...
    ...
public:
    bst();
    void insert(N data);
    treeNode<N> *search(N data);
    void inorder();
    vector<N> vectorize();
    int node_count();
```



```
template <typename T>
struct treeNode{
    T data;
    treeNode *left;
    treeNode *right;
};
```

C++ Example

- Using classes in C++, create a BST class and perform or find:
 - Insertion
 - Searching
 - Traversal
 - Vectorization
 - Size of tree (# of nodes)
 - Find height of the tree

```
template <class N>
class bst{
private:
    ...
    ...
    ...
public:
    bst();
    void insert(N data);
    treeNode<N> *search(N data);
    void inorder();
    vector<N> vectorize();
    int node_count();
    int height();
```

```
template <typename T>
struct treenode{
    T data;
    treenode *left;
    treenode *right;
};
```

C++ Example

- Using classes in C++, create a BST class and perform or find:
 - Insertion
 - Searching
 - Traversal
 - Vectorization
 - Size of tree (# of nodes)
 - Find height of the tree

```
template <class N>
class bst{
private:
    ...
    ...
    ...
public:
    bst();
    void insert(N data);
    treenode<N> *search(N data);
    void inorder();
    vector<N> vectorize();
    int node_count();
    int height();
    void print();
```

```
template <typename T>
struct treenode{
    T data;
    treenode *left;
    treenode *right;
};
```

C++ Example

- Using classes in C++, create a BST class and perform or find:
 - Insertion
 - Searching
 - Traversal
 - Vectorization
 - Size of tree (# of nodes)
 - Find height of the tree
 - Deletion of tree

```
template <class N>
class bst{
private:
    ...
    ...
    ...
public:
    bst();
    void insert(N data);
    treenode<N> *search(N data);
    void inorder();
    vector<N> vectorize();
    int node_count();
    int height();
    void print();
    ~bst();
};
```

```
template <typename T>
struct treeNode{
    T data;
    treeNode *left;
    treeNode *right;
};
```

C++ Example

- Using classes in C++, create a BST class and perform or find:

```
template <class N>
class bst{
private:
    typedef treeNode<N> node;
    node *root;
```

```
template <typename T>
struct treeNode{
    T data;
    treeNode *left;
    treeNode *right;
};
```

C++ Example

- Insertion

```
template <class N>
class bst{
private:
    typedef treeNode<N> node;
    node *root;

    void insert(N data, node **cursor);
```

```
template <typename T>
struct treeNode{
    T data;
    treeNode *left;
    treeNode *right;
};
```

C++ Example

- Insertion
- Searching

```
template <class N>
class bst{
private:
    typedef treeNode<N> node;
    node *root;

    void insert(N data, node **cursor);
    node *search(N key, node *cursor);
```

```
template <typename T>
struct treeNode{
    T data;
    treeNode *left;
    treeNode *right;
};
```

C++ Example

- Insertion
- Searching
- Traversal

```
template <class N>
class bst{
private:
    typedef treeNode<N> node;
    node *root;

    void insert(N data, node **cursor);
    node *search(N key, node *cursor);
    void inorder(node *cursor);
```

```
template <typename T>
struct treeNode{
    T data;
    treeNode *left;
    treeNode *right;
};
```

C++ Example

- Insertion
- Searching
- Traversal
- Vectorization

```
template <class N>
class bst{
private:
    typedef treeNode<N> node;
    node *root;

    void insert(N data, node **cursor);
    node *search(N key, node *cursor);
    void inorder(node *cursor);
    vector<N> vectorize(node *cursor, vector<N> &v);
```



```
template <typename T>
struct treenode{
    T data;
    treenode *left;
    treenode *right;
};
```

C++ Example

- Insertion
- Searching
- Traversal
- Vectorization
- Size of tree (# of nodes)

```
template <class N>
class bst{
private:
    typedef treenode<N> node;
    node *root;

    void insert(N data, node **cursor);
    node *search(N key, node *cursor);
    void inorder(node *cursor);
    vector<N> vectorize(node *cursor, vector<N> &v);
    int countnodes(node *cursor);
    void print(node *cursor, int depth);
```

```
template <typename T>
struct treeNode{
    T data;
    treeNode *left;
    treeNode *right;
};
```

C++ Example

- Insertion
- Searching
- Traversal
- Vectorization
- Size of tree (# of nodes)
- Find height of the tree

```
template <class N>
class bst{
private:
    typedef treeNode<N> node;
    node *root;

    void insert(N data, node **cursor);
    node *search(N key, node *cursor);
    void inorder(node *cursor);
    vector<N> vectorize(node *cursor, vector<N> &v);
    int countnodes(node *cursor);
    void print(node *cursor, int depth);
    int height(node *cursor);
```

```
template <typename T>
struct treenode{
    T data;
    treenode *left;
    treenode *right;
};
```

C++ Example

- Insertion
- Searching
- Traversal
- Vectorization
- Size of tree (# of nodes)
- Find height of the tree
- Deletion of tree

```
template <class N>
class bst{
private:
    typedef treenode<N> node;
    node *root;

    void insert(N data, node **cursor);
    node *search(N key, node *cursor);
    void inorder(node *cursor);
    vector<N> vectorize(node *cursor, vector<N> &v);
    int countnodes(node *cursor);
    void print(node *cursor, int depth);
    int height(node *cursor);
    void delete_tree(node *cursor);
```

```
template <typename T>
struct treenode{
    T data;
    treenode *left;
    treenode *right;
};
```

C++ Example

- Insertion
- Searching
- Traversal
- Vectorization
- Size of tree (# of nodes)
- Find height of the tree
- Deletion of tree

```
template <class N>
class bst{
private:
    typedef treenode<N> node;
    node *root;

    void insert(N data, node **cursor);
    node *search(N key, node *cursor);
    void inorder(node *cursor);
    vector<N> vectorize(node *cursor, vector<N> &v);
    int countnodes(node *cursor);
    void print(node *cursor, int depth);
    int height(node *cursor);
    void delete_tree(node *cursor);
```

```
template <typename T>
struct treenode{
    T data;
    treenode *left;
    treenode *right;
};
```

C++ Example

- Insertion
- Searching
- Traversal
- Vectorization
- Size of tree (# of nodes)
- Find height of the tree
- Deletion of tree

```
template <class N>
class bst{
private:
    typedef treenode<N> node;
    node *root;

    void insert(N data, node **cursor);
    node *search(N key, node *cursor);
    void inorder(node *cursor);
    vector<N> vectorize(node *cursor, vector<N> &v);
    int countnodes(node *cursor);
    void print(node *cursor, int depth);
    int height(node *cursor);
    void delete_tree(node *cursor);

public:
```

```
template <typename T>
struct treeNode{
    T data;
    treeNode *left;
    treeNode *right;
};
```

C++ Example

- Insertion
- Searching
- Traversal
- Vectorization
- Size of tree (# of nodes)
- Find height of the tree
- Deletion of tree

```
template <class N>
class bst{
private:
    typedef treeNode<N> node;
    node *root;

    void insert(N data, node **cursor);
    node *search(N key, node *cursor);
    void inorder(node *cursor);
    vector<N> vectorize(node *cursor, vector<N> &v);
    int countnodes(node *cursor);
    void print(node *cursor, int depth);
    int height(node *cursor);
    void delete_tree(node *cursor);

public:
    ...
```

```
template <typename T>
struct treenode{
    T data;
    treenode *left;
    treenode *right;
};
```

C++ Example

- Insertion
- Searching
- Traversal
- Vectorization
- Size of tree (# of nodes)
- Find height of the tree
- Deletion of tree

```
template <class N>
class bst{
private:
    typedef treenode<N> node;
    node *root;

    void insert(N data, node **cursor);
    node *search(N key, node *cursor);
    void inorder(node *cursor);
    vector<N> vectorize(node *cursor, vector<N> &v);
    int countnodes(node *cursor);
    void print(node *cursor, int depth);
    int height(node *cursor);
    void delete_tree(node *cursor);

public:
    ...
    ...
```

C++ Example

C++ Example

```
#include <iostream>
#include "bst.hpp"

using namespace std;

int main(){
    bst <int> tree1;
    cout<<"Building a Binary Search Tree"<<endl;

    tree1.insert(45);
    tree1.insert(50);
    tree1.insert(35);
    tree1.insert(30);
    tree1.insert(70);
    tree1.insert(20);
    tree1.insert(40);
    tree1.insert(80);
    tree1.insert(60);

    cout<<"Total number of nodes in this tree: ";
    cout<<tree1.node_count()<<endl;
    tree1.inorder();

    cout<<endl;
    tree1.print();
    cout<<"The tree height is: "<<tree1.height();
    cout<<endl;

    vector <int> v = tree1.vectorize();
    cout<<"Vectorized in order this is:"<<endl;
    for (auto it= v.begin(); it != v.end(); ++it)
        cout<<*it<<" , ";
    return 0;
}
```

Happy Fall Break

- Take survey: <https://surveys.illinois.edu/sec/1742038613>
- Stay warm and travel safe!

Happy Fall Break

- Take survey: <https://surveys.illinois.edu/sec/1742038613>
- Stay warm and travel safe!

