

Binary Search Tree (1/3): Search and Insertion

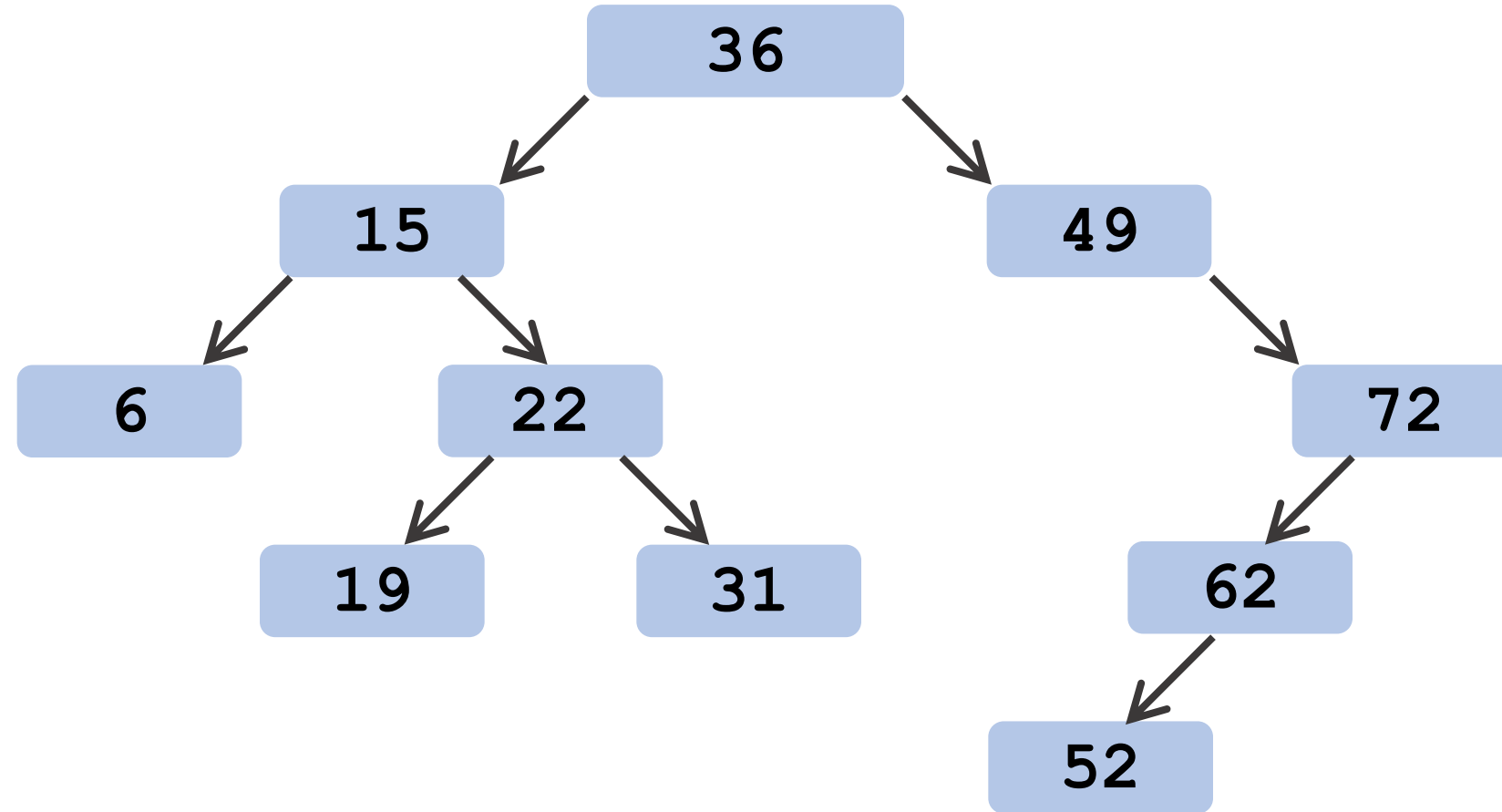
Shusen Wang

Stevens Institute of Technology

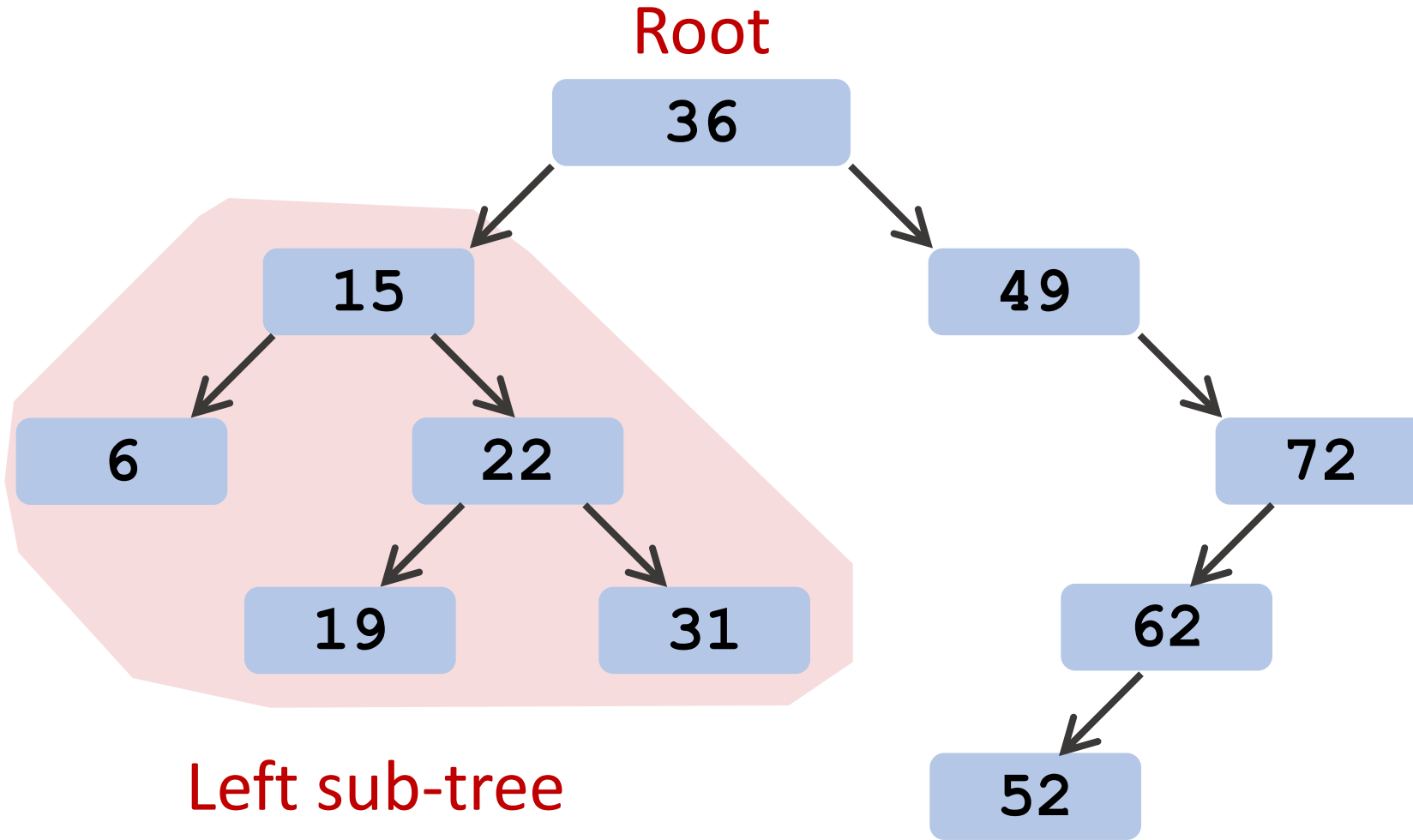
<http://wangshusen.github.io/>

Binary Search Tree

Root



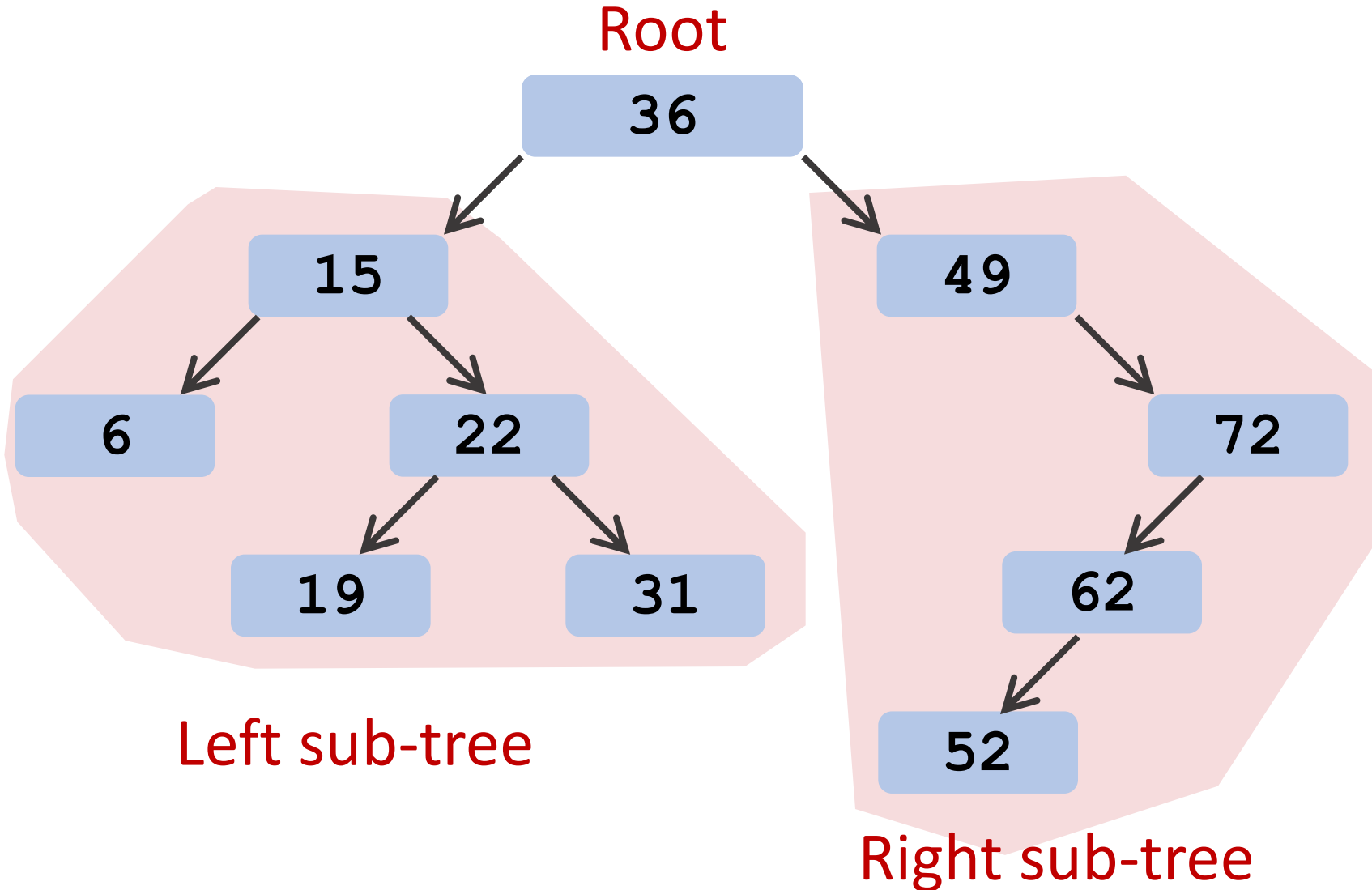
Binary Search Tree



Property:

- All the keys in the **left sub-tree** are **smaller** than the **root's** key.

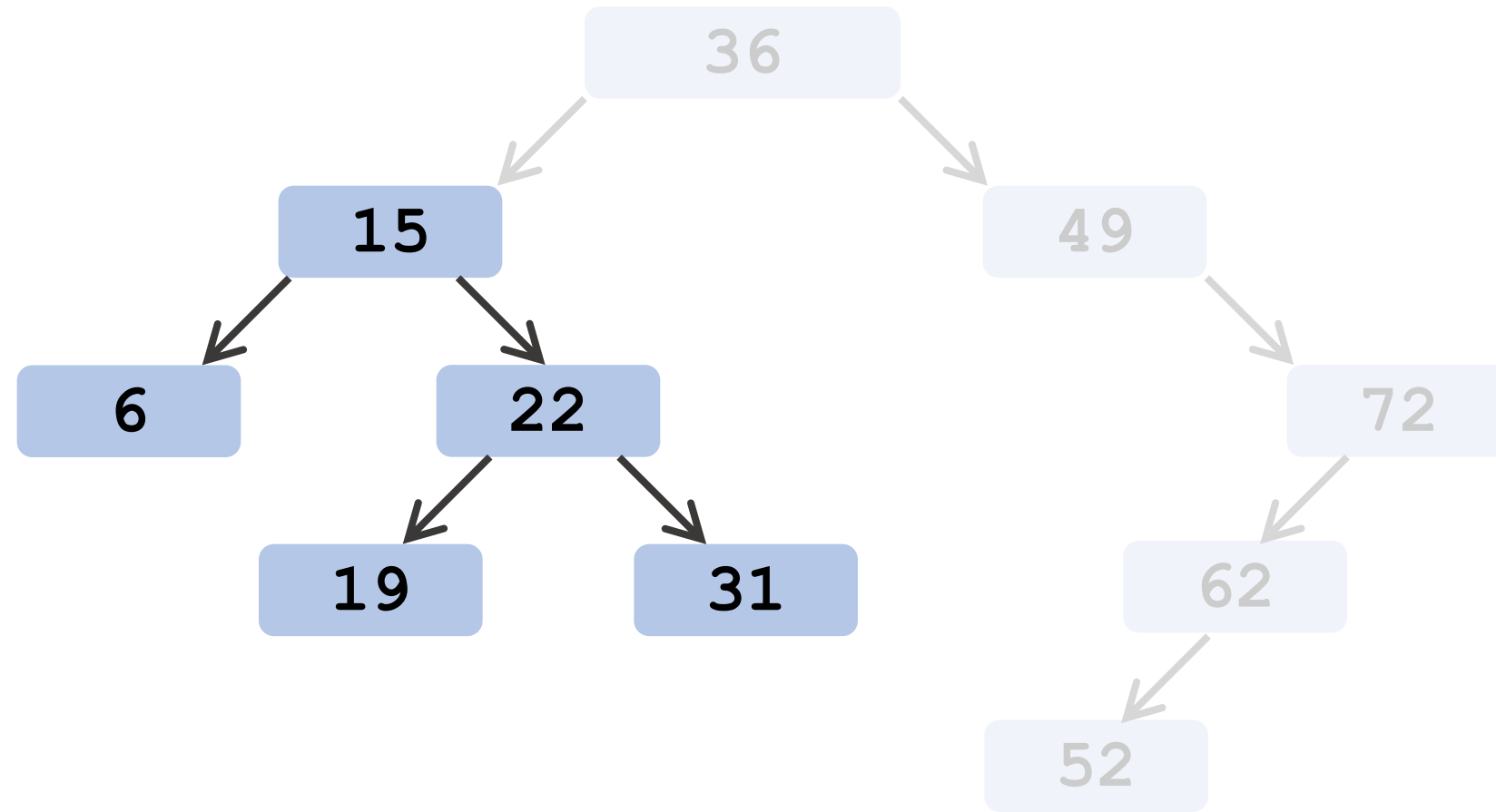
Binary Search Tree



Property:

- All the keys in the **left sub-tree** are **smaller** than the **root's** key.
- All the keys in the **right sub-tree** are **greater** than the **root's** key.

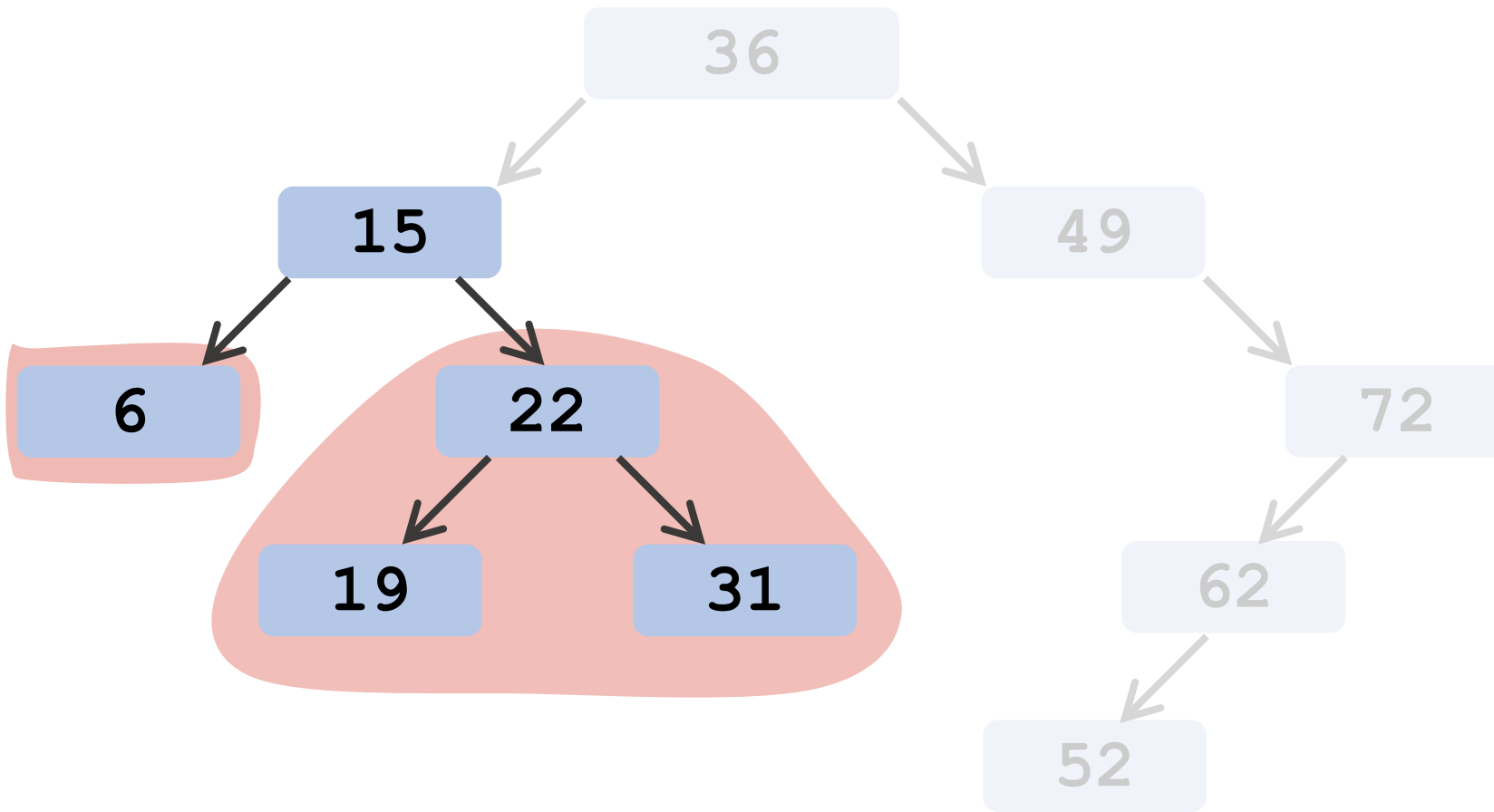
Binary Search Tree



Property:

- All the keys in the **left sub-tree** are **smaller** than the **root's** key.
- All the keys in the **right sub-tree** are **greater** than the **root's** key.

Binary Search Tree



Property:

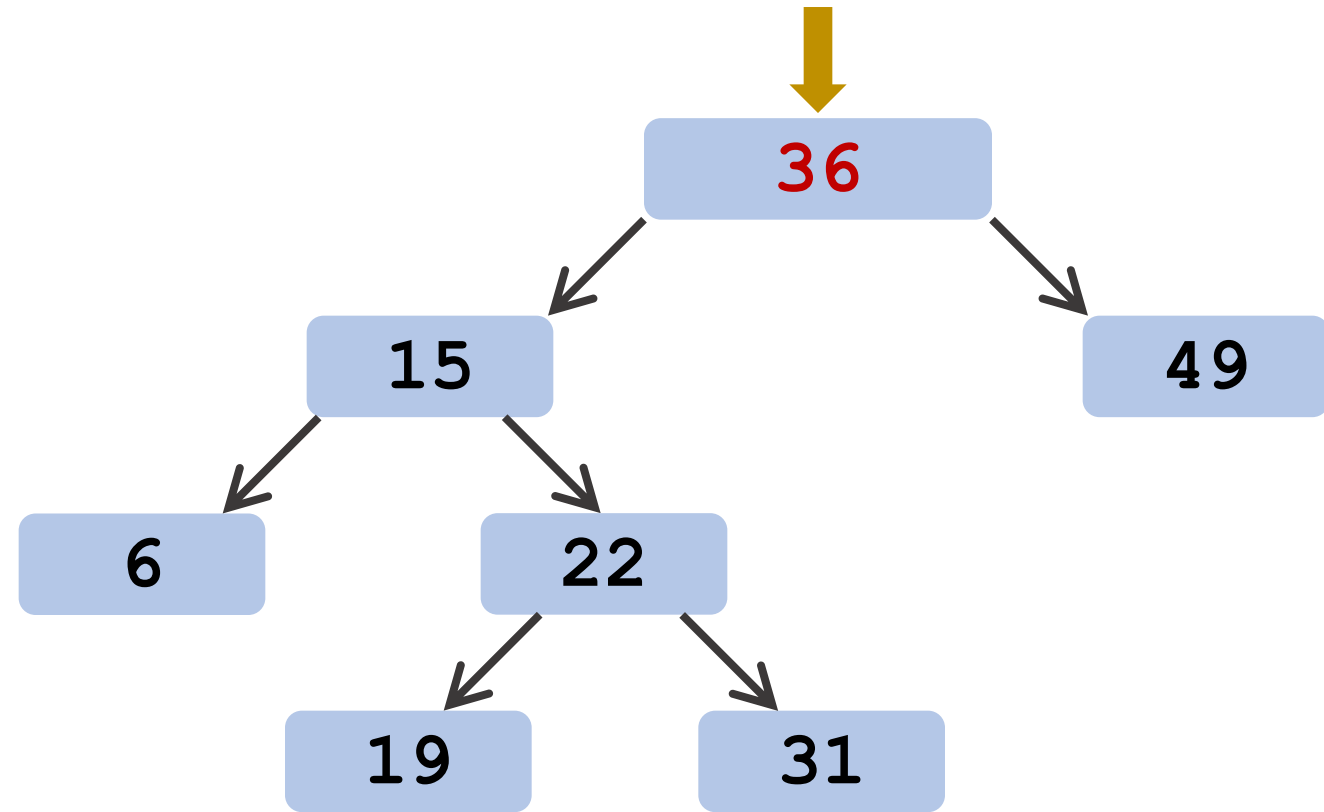
- All the keys in the **left sub-tree** are **smaller** than the **root's** key.
- All the keys in the **right sub-tree** are **greater** than the **root's** key.

Search

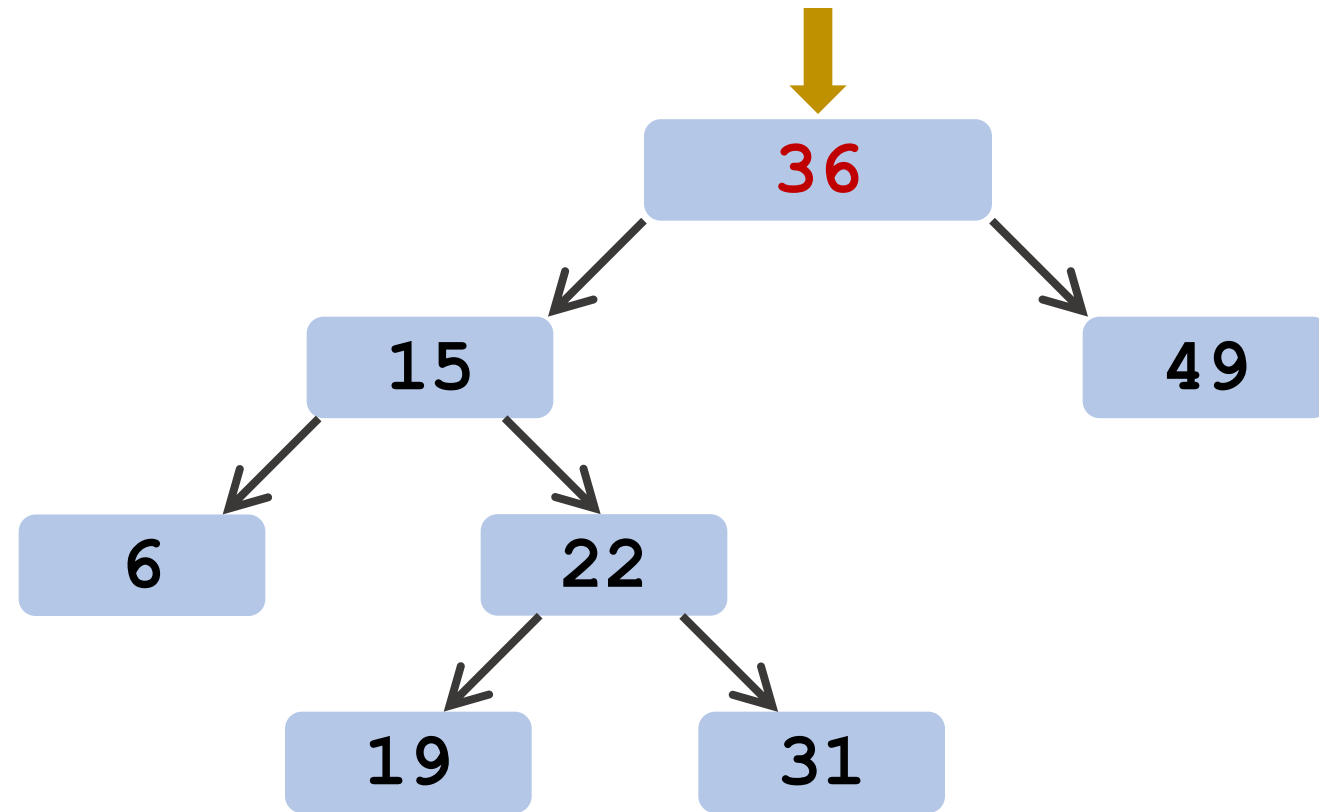
Search

- **Inputs:** **root** (of the tree) and **key** (to be matched).
- **Goal:** find the vertex which matches the input key.
- **Output:** **the vertex** (if found) or **NULL** (if not found).
- **Time complexity:** depth of the tree.

Search “**key=36**”



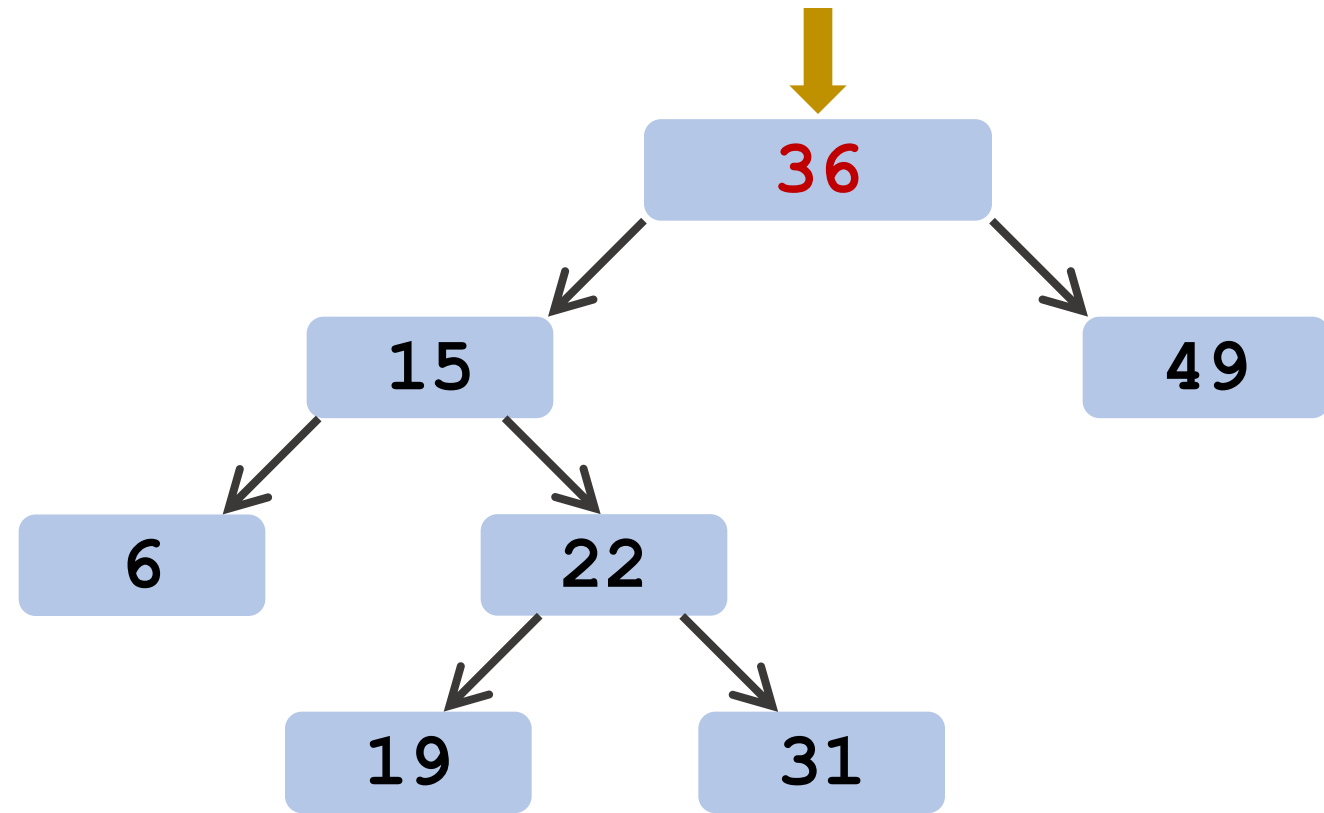
Search “**key=36**”



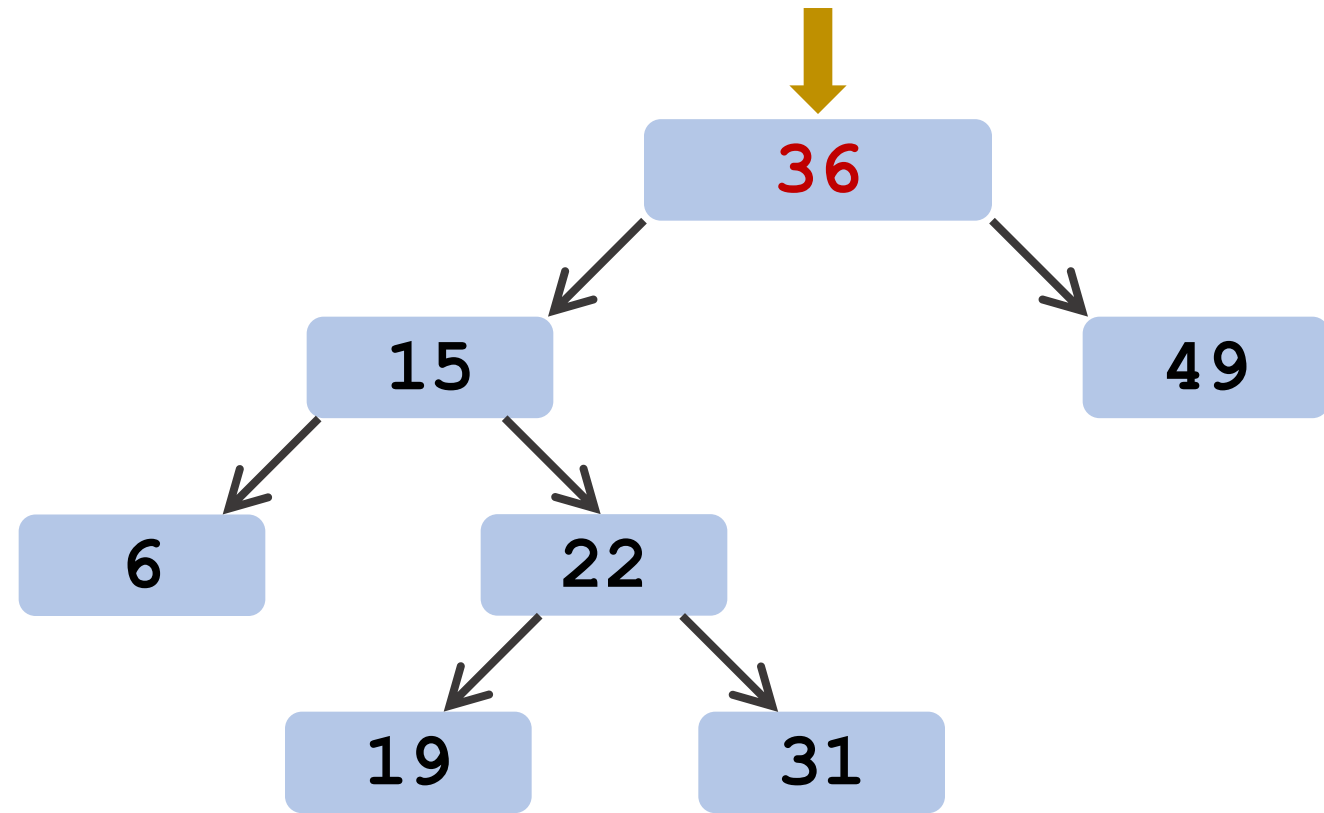
1. **key == 36** ➔ Found!

2. Return the root **36**

Search “**key=22**”



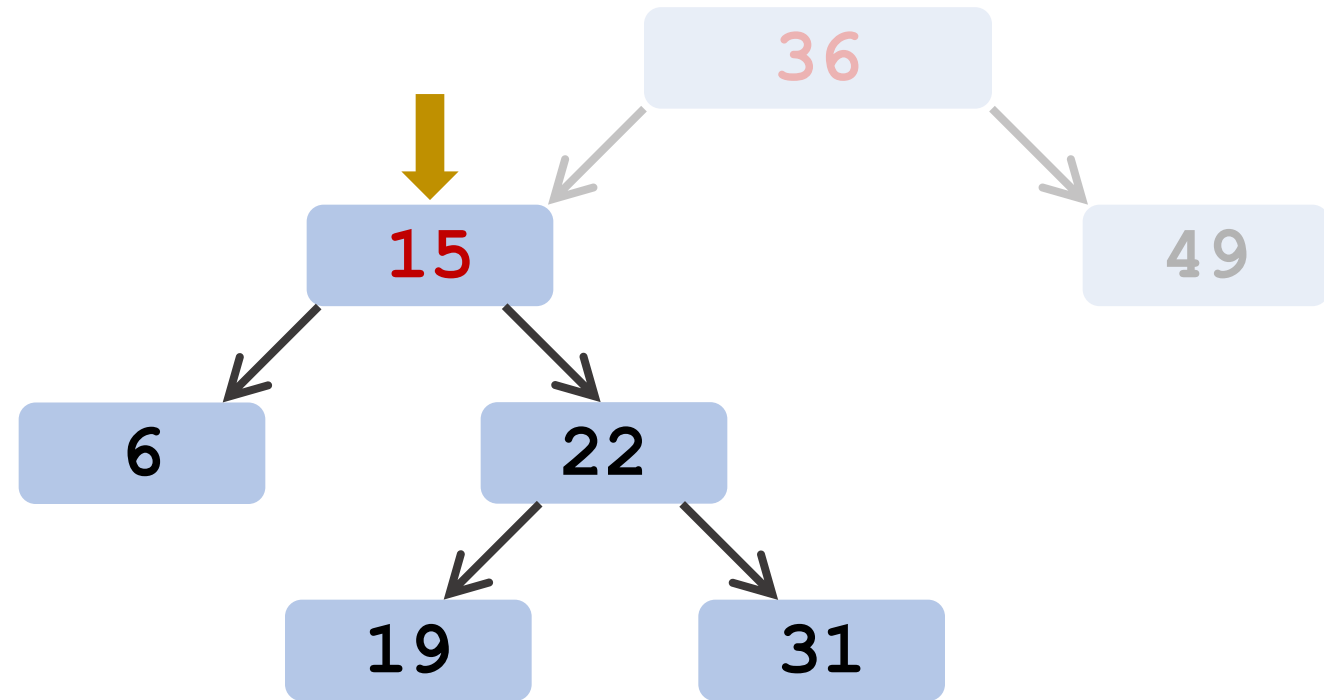
Search “key=22”



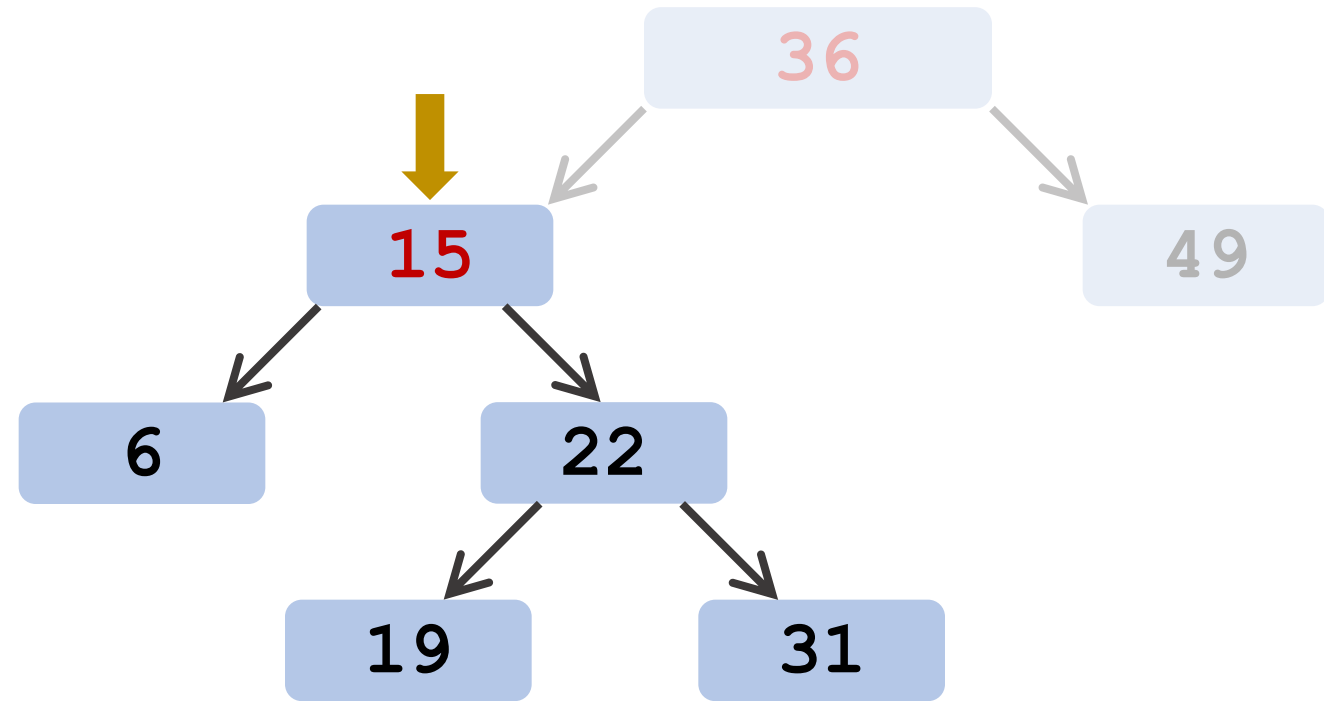
1. $\text{key} < 36 \Rightarrow$ Go to left.

Search “key=22”

1. $\text{key} < 36 \Rightarrow$ Go to left.



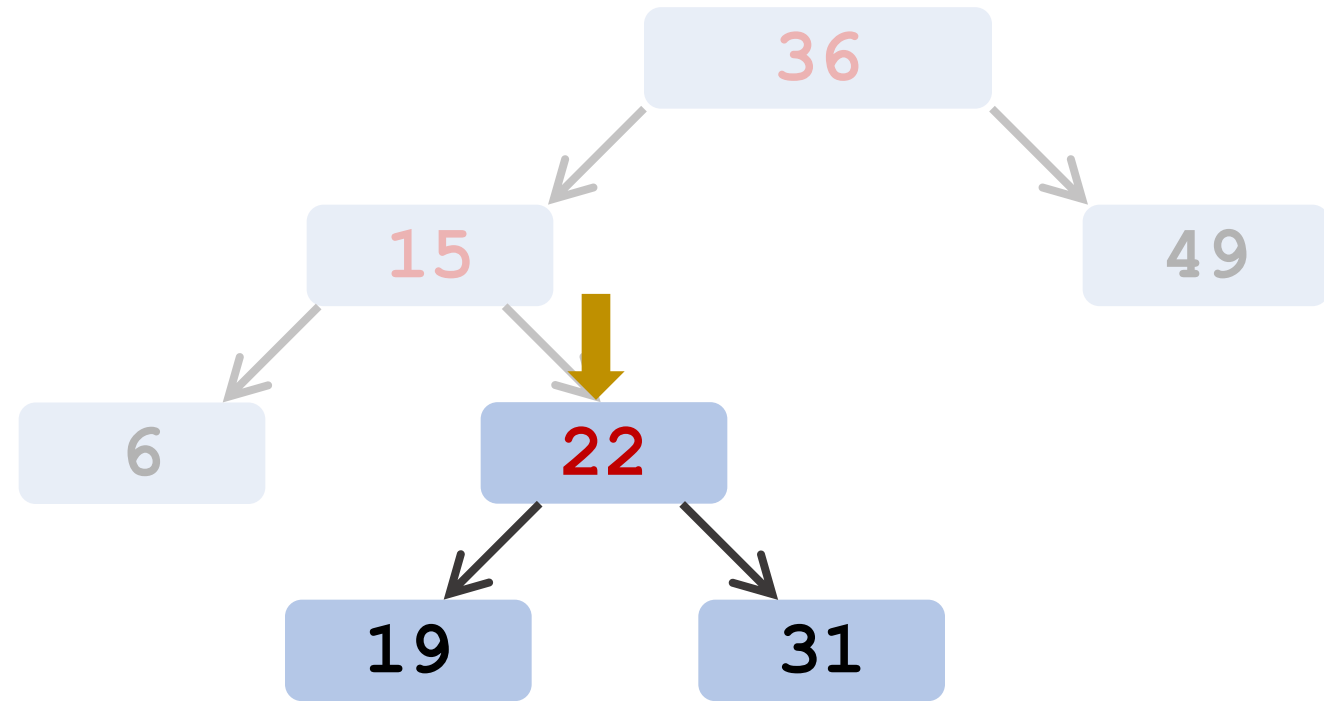
Search “key=22”



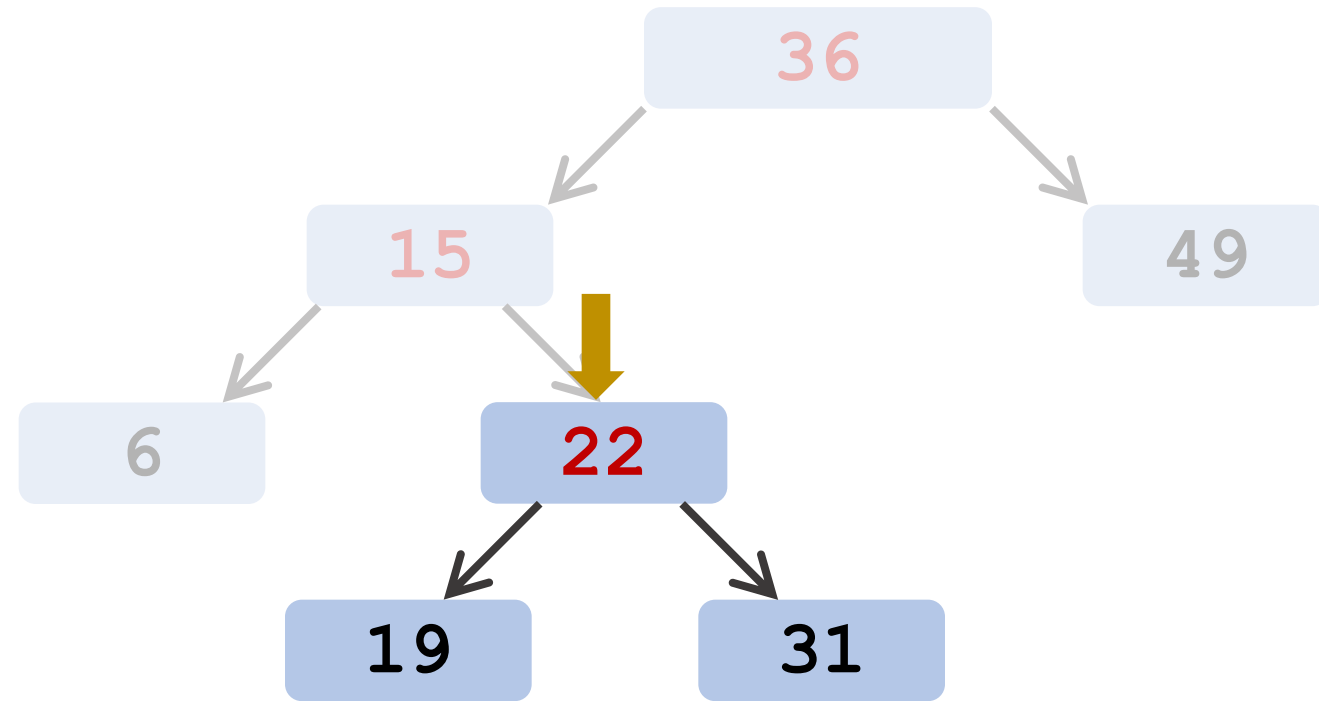
1. $\text{key} < 36 \Rightarrow$ Go to left.
2. $\text{key} > 15 \Rightarrow$ Go to right.

Search “key=22”

1. $\text{key} < 36 \Rightarrow$ Go to left.
2. $\text{key} > 15 \Rightarrow$ Go to right.

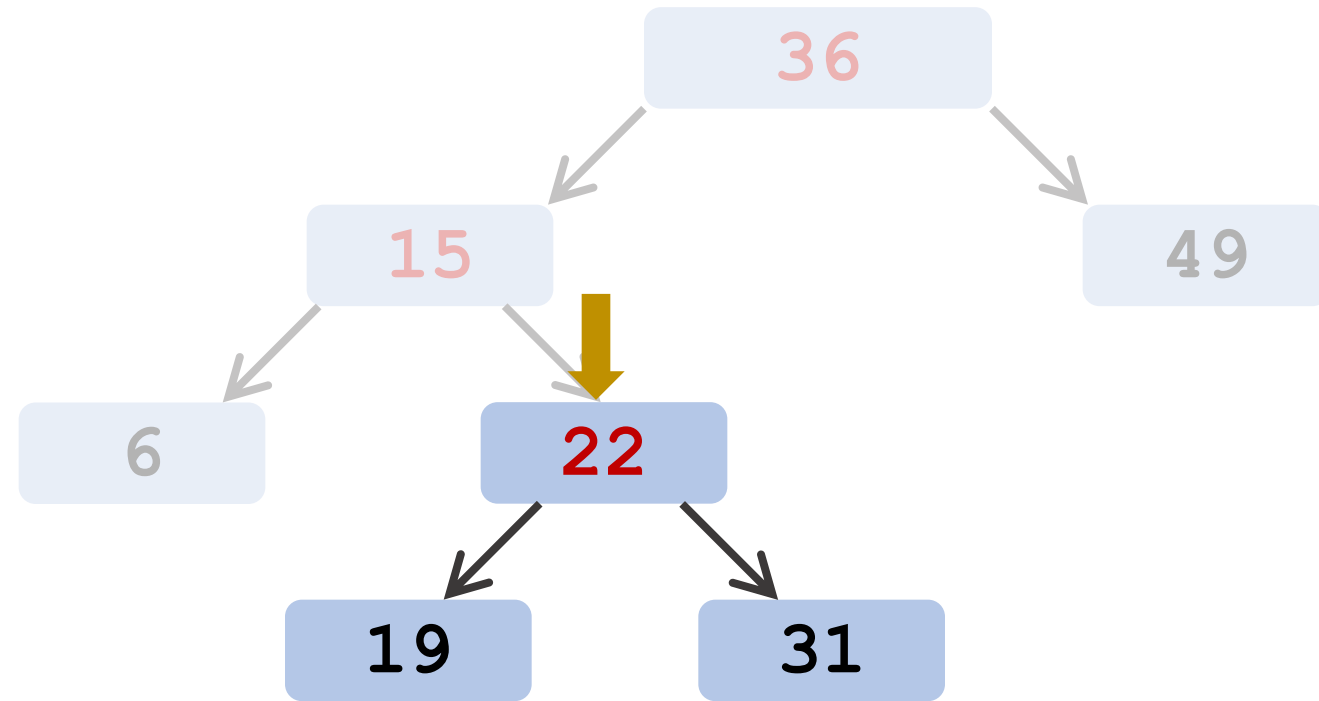


Search “key=22”



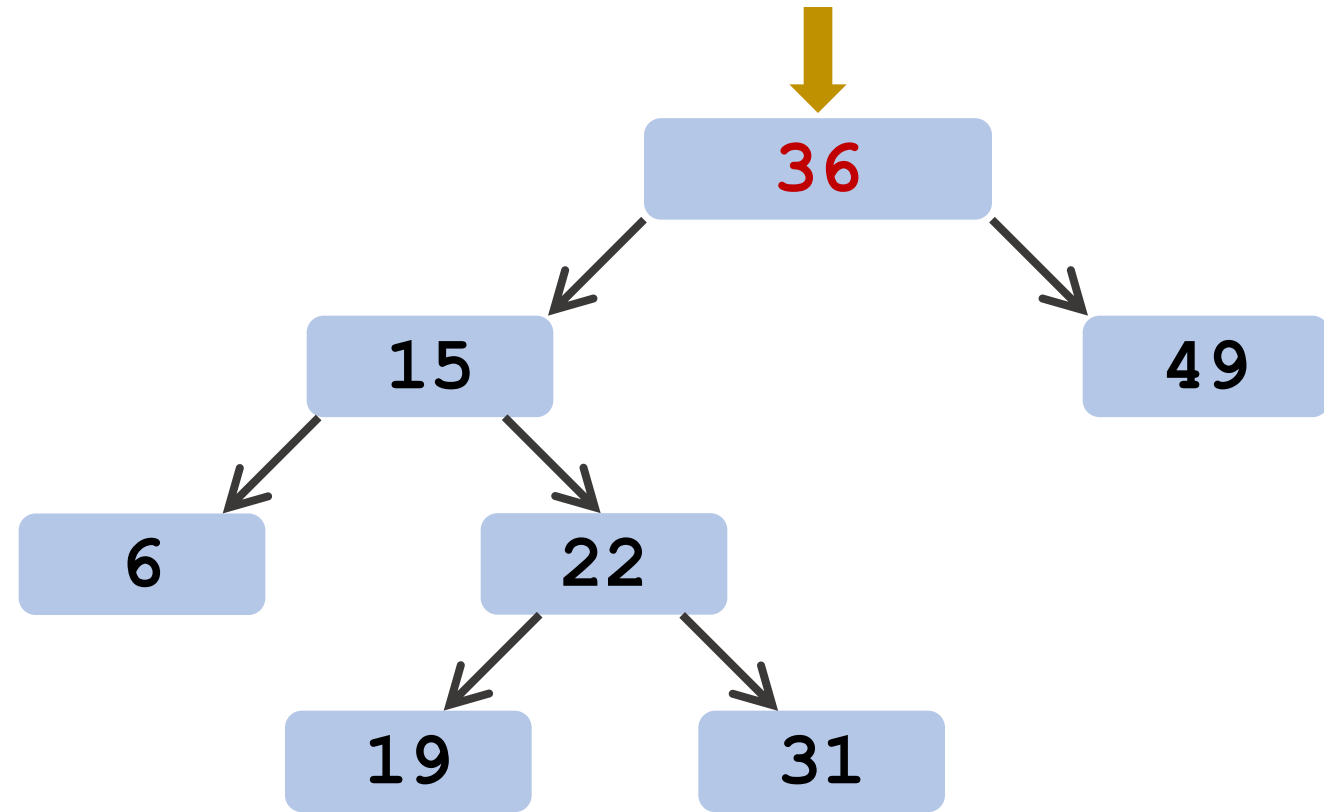
1. $\text{key} < 36 \Rightarrow$ Go to left.
2. $\text{key} > 15 \Rightarrow$ Go to right.
3. $\text{key} == 22 \Rightarrow$ Found!

Search “**key=22**”

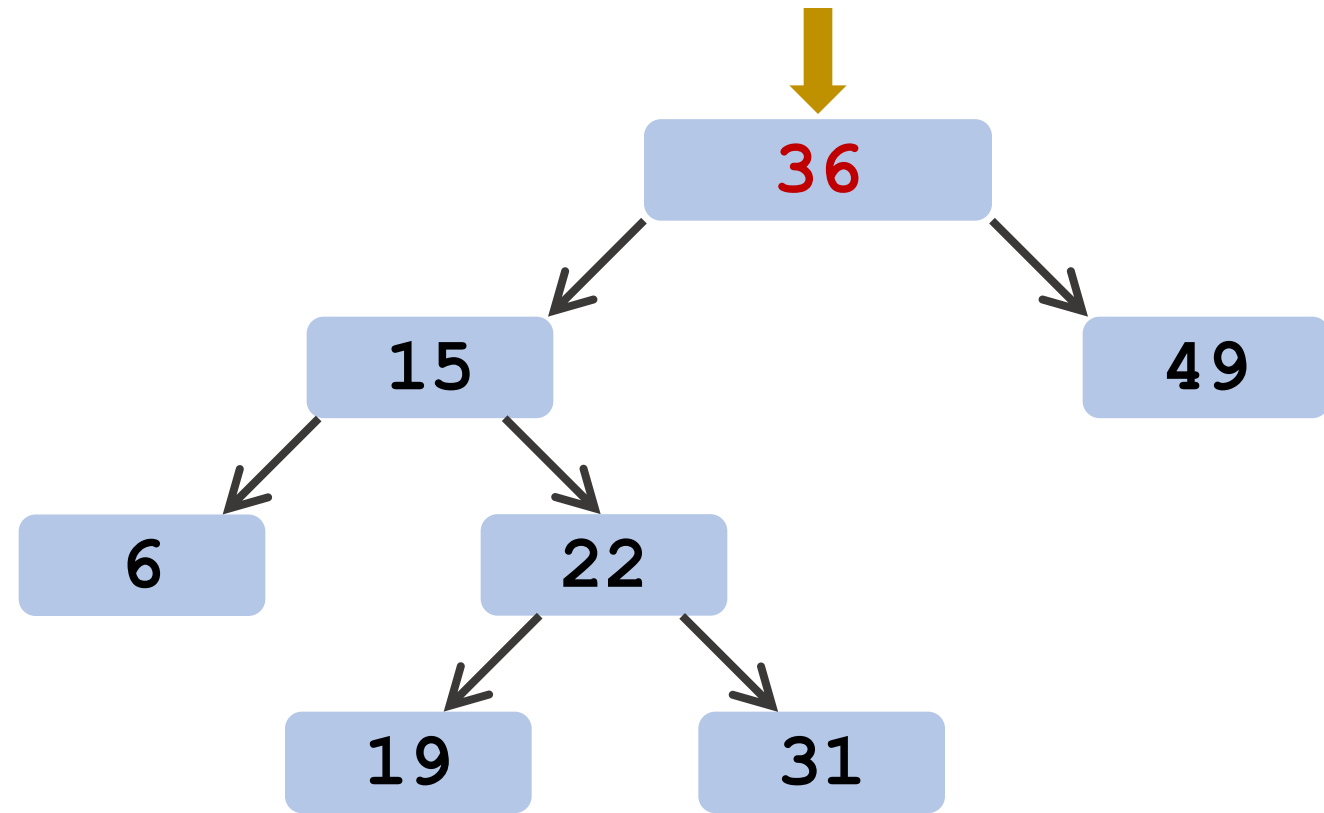


1. $\text{key} < 36 \rightarrow$ Go to left.
2. $\text{key} > 15 \rightarrow$ Go to right.
3. $\text{key} == 22 \rightarrow$ Found!
4. Return the vertex **22**

Search “**key=40**”



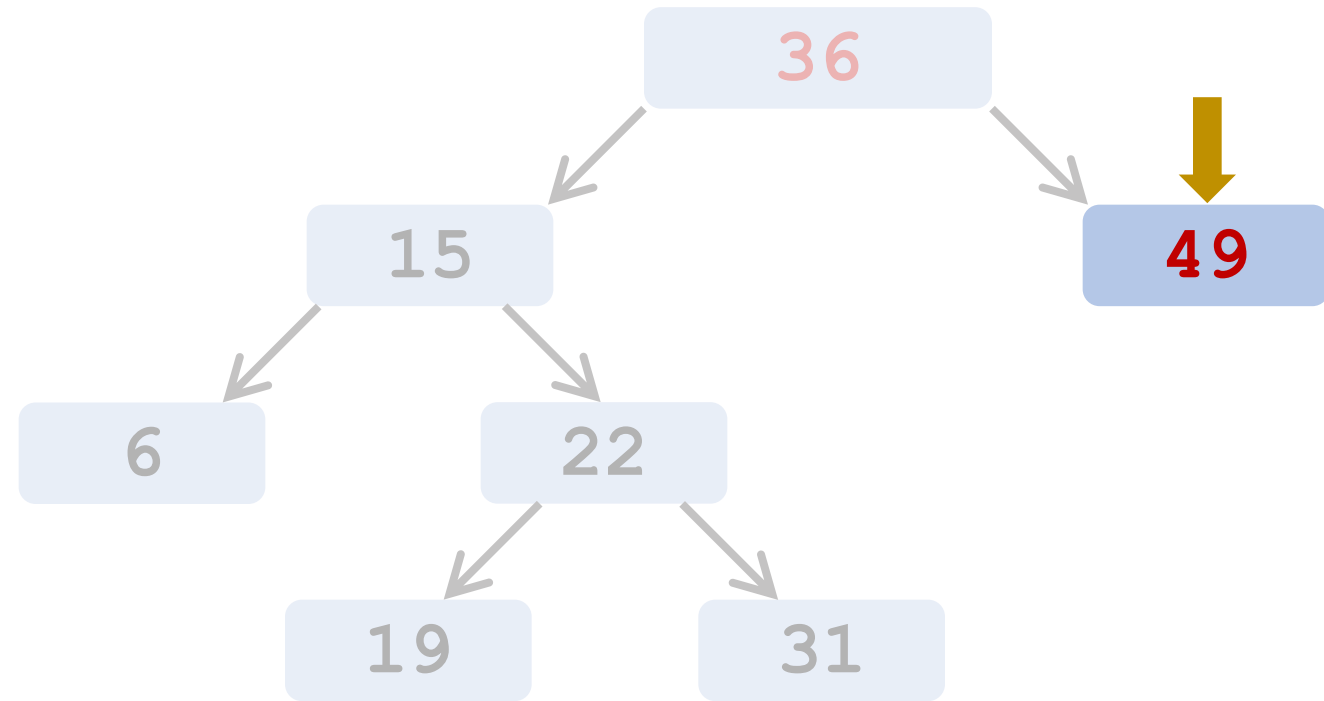
Search “key=40”



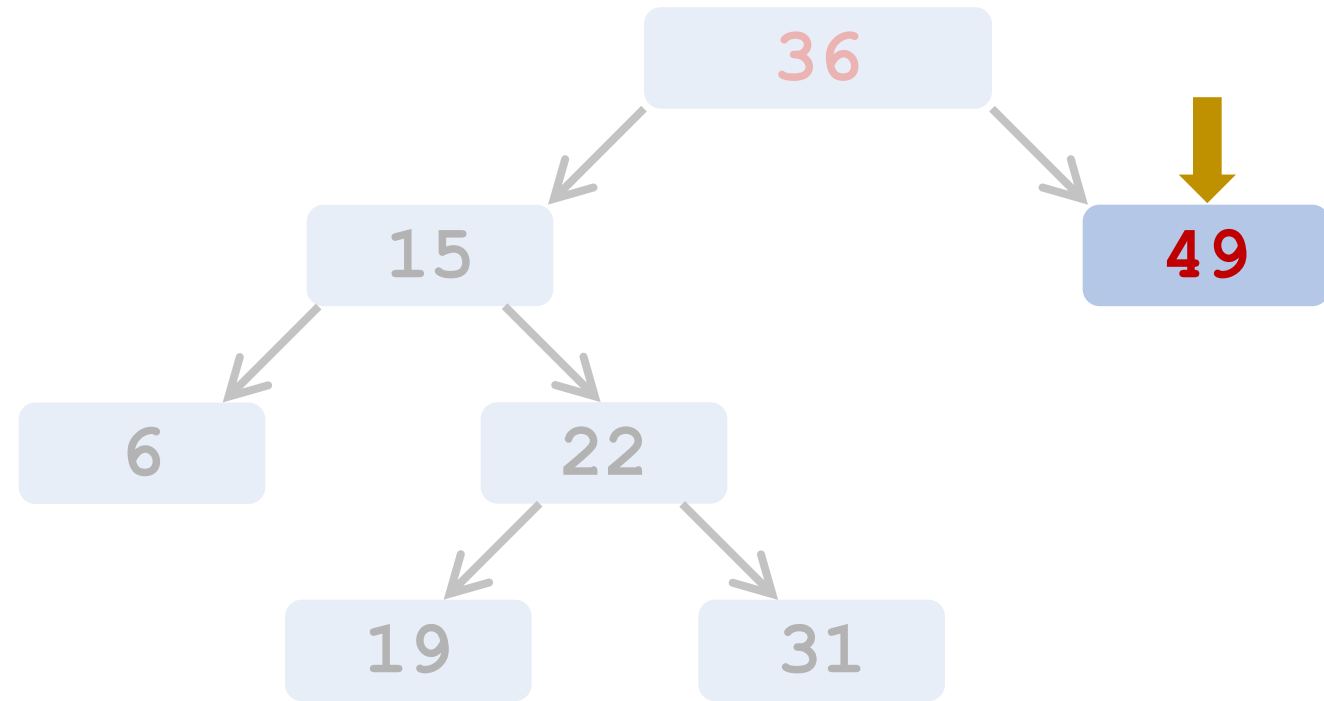
1. $\text{key} > 36 \Rightarrow$ Go to right.

Search “key=40”

1. $\text{key} > 36 \Rightarrow$ Go to right.



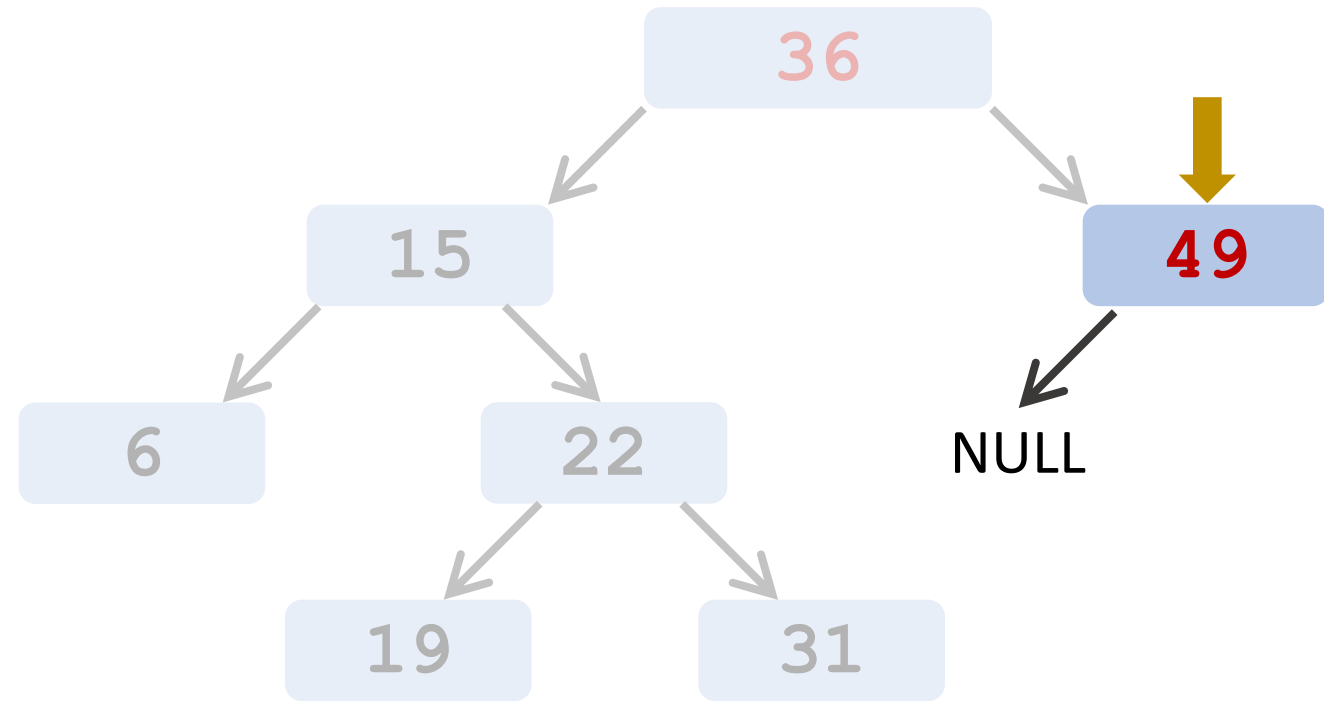
Search “key=40”



1. $\text{key} > 36 \Rightarrow$ Go to right.

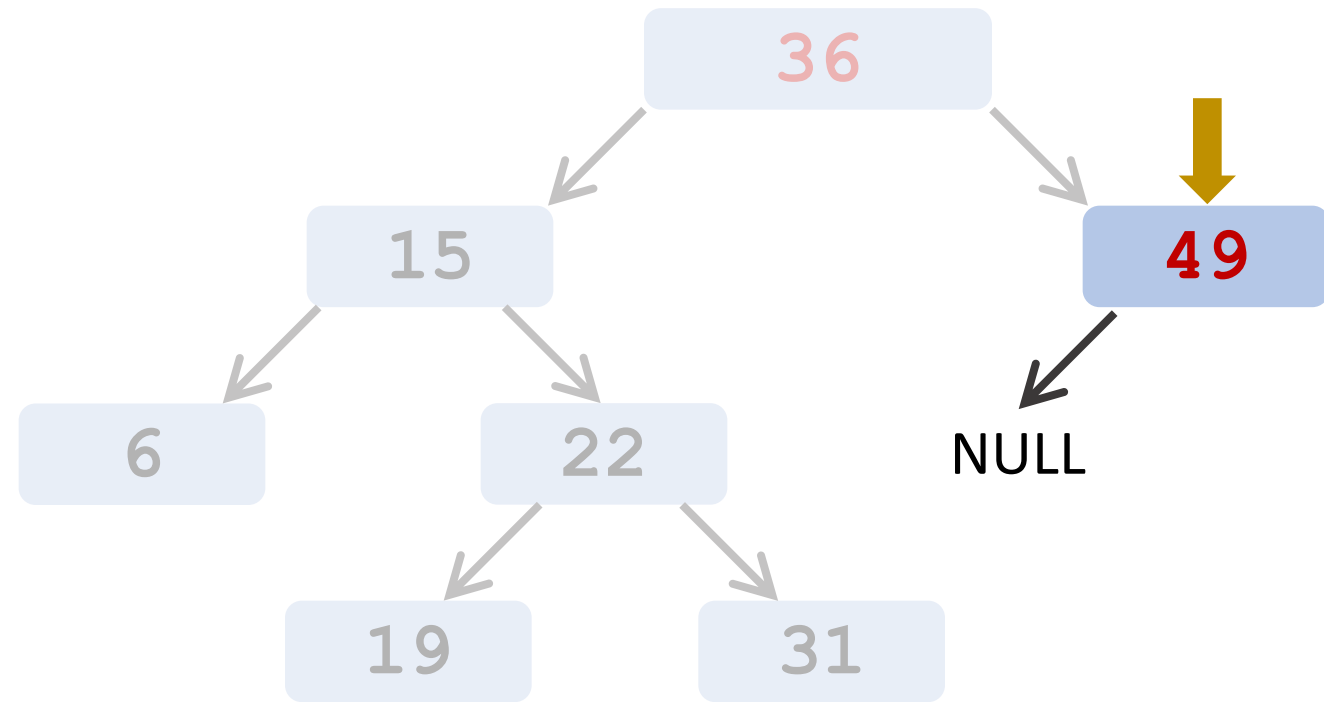
2. $\text{key} < 49 \Rightarrow$ Go to left.

Search “key=40”



1. $\text{key} > 36 \Rightarrow$ Go to right.
2. $\text{key} < 49 \Rightarrow$ Go to left.
3. The left child is NULL.

Search “key=40”




1. $\text{key} > 36 \Rightarrow$ Go to right.
2. $\text{key} < 49 \Rightarrow$ Go to left.
3. The left child is NULL.
4. Return NULL.

```
struct vertex* search(struct vertex* root, int key) {  
    // empty tree or the key is present at root  
    if (root == NULL || root->key == key)  
        return root;  
    // key is in the right subtree  
    if (key > root->key)  
        return search(root->right, key);  
    // key is in the left subtree  
    else  
        return search(root->left, key);  
}
```



```
struct vertex* search(struct vertex* root, int key) {  
    // empty tree or the key is present at root  
    ➡ if (root == NULL || root->key == key)  
        return root;  
    // key is in the right subtree  
    if (key > root->key)  
        return search(root->right, key);  
    // key is in the left subtree  
    else  
        return search(root->left, key);  
}
```

```
struct vertex* search(struct vertex* root, int key) {  
    // empty tree or the key is present at root  
    if (root == NULL || root->key == key)  
        return root;  
    // key is in the right subtree  
    ➡ if (key > root->key)  
        return search(root->right, key);  
    // key is in the left subtree  
    else  
        return search(root->left, key);  
}
```

```
struct vertex* search(struct vertex* root, int key) {  
    // empty tree or the key is present at root  
    if (root == NULL || root->key == key)  
        return root;  
    // key is in the right subtree  
    if (key > root->key)  
        return search(root->right, key);  
    // key is in the left subtree  
     else  
        return search(root->left, key);  
}
```

Insertion

Insert

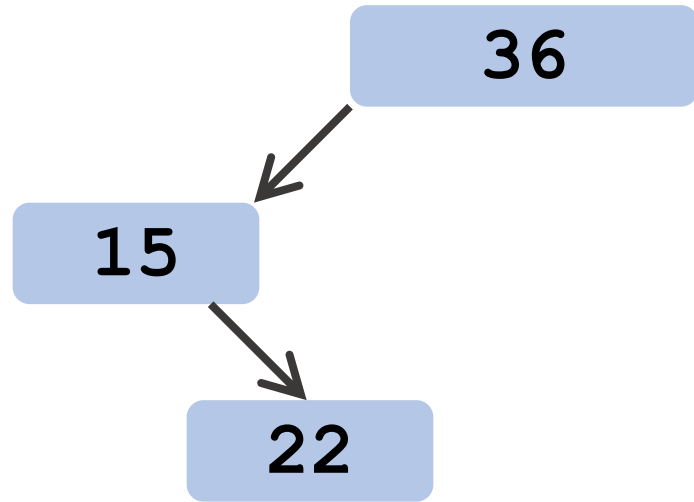
- **Inputs:** **root** (of the tree) and **key** (to be matched).
- **Goal:** create a new vertex and insert it into the correct position.
- **Time complexity:** depth of the tree.

Insert

Question: What if the tree is empty (i.e., `root==NULL`)?

- Create a new vertex and make it the root.
- Return the root.

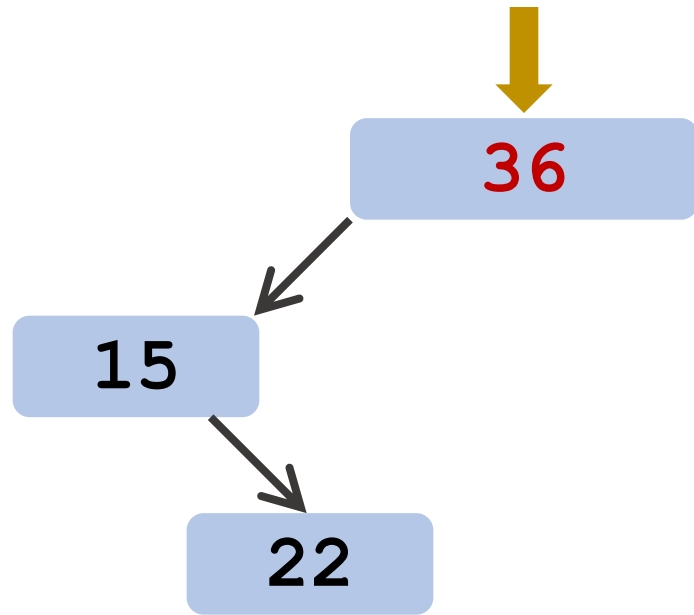
Insert “key=49”



1. Create new vertex

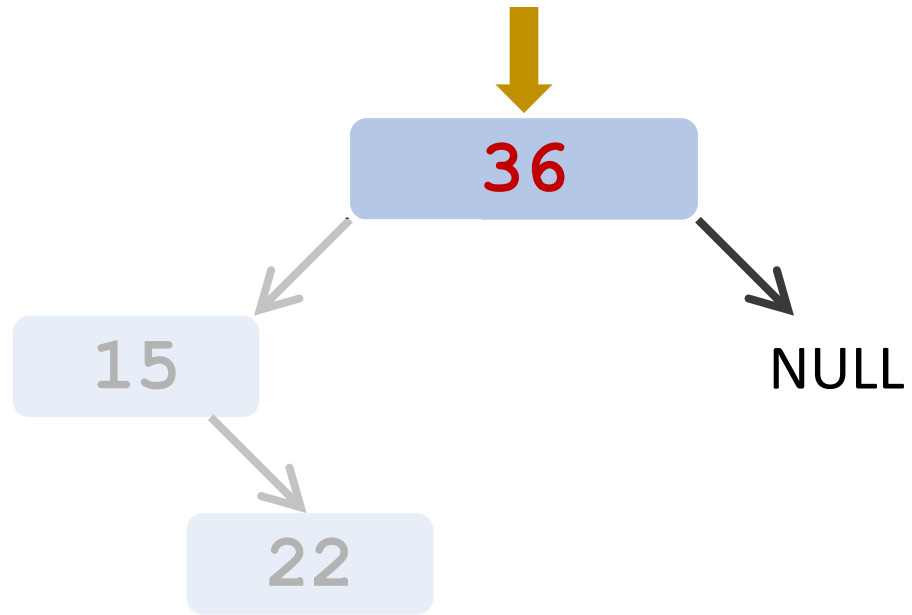
49

Insert “key=49”



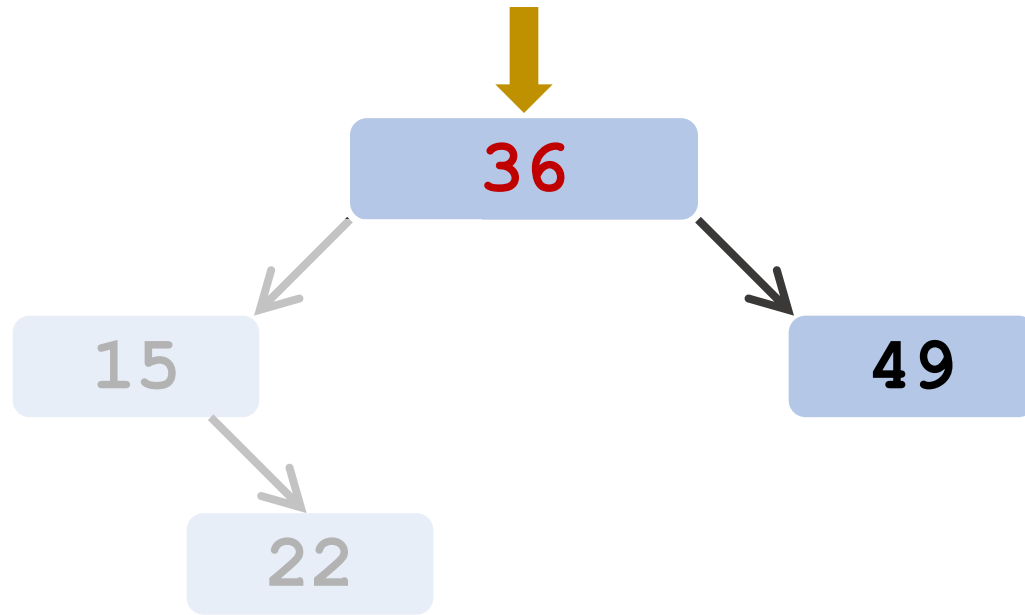
1. Create new vertex **49**
2. $\text{key} > 36 \Rightarrow$ Go to right.

Insert “key=49”



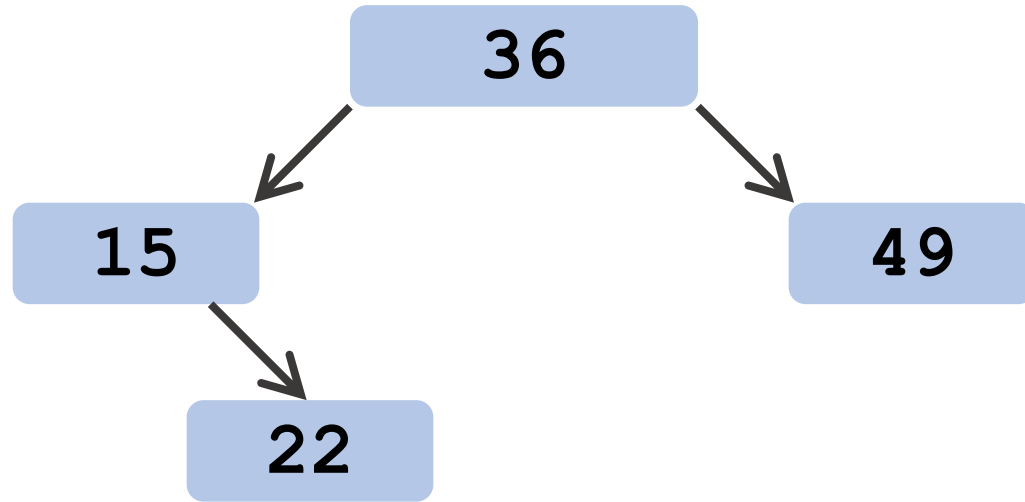
1. Create new vertex **49**
2. **key** > **36** ➔ Go to right.
3. Right child is NULL

Insert “key=49”



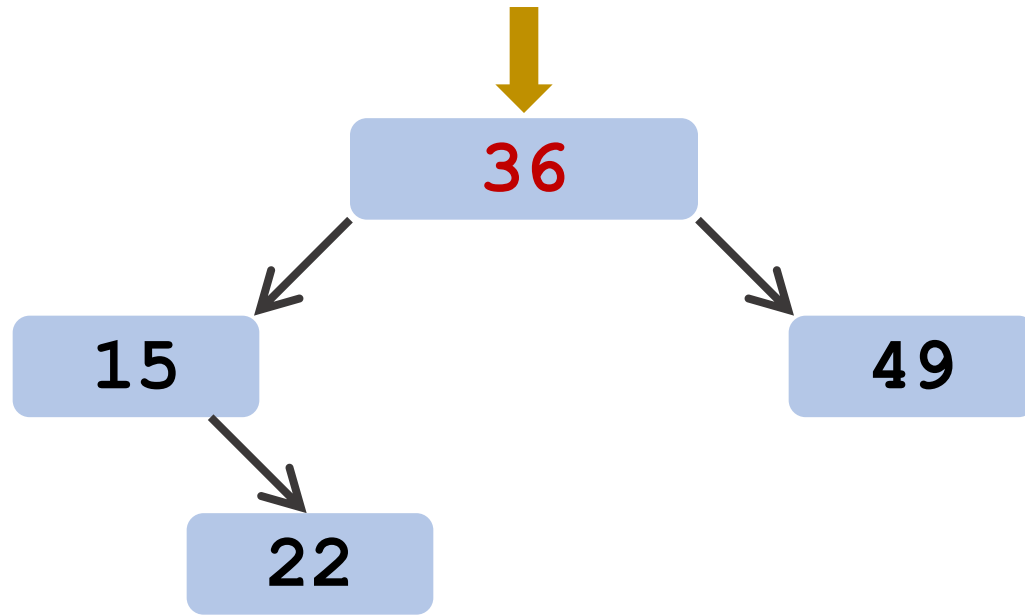
1. Create new vertex **49**
2. **key** > **36** → Go to right.
3. Right child is NULL → Make “49” the right child.

Insert “key=31”



1. Create new vertex **31**

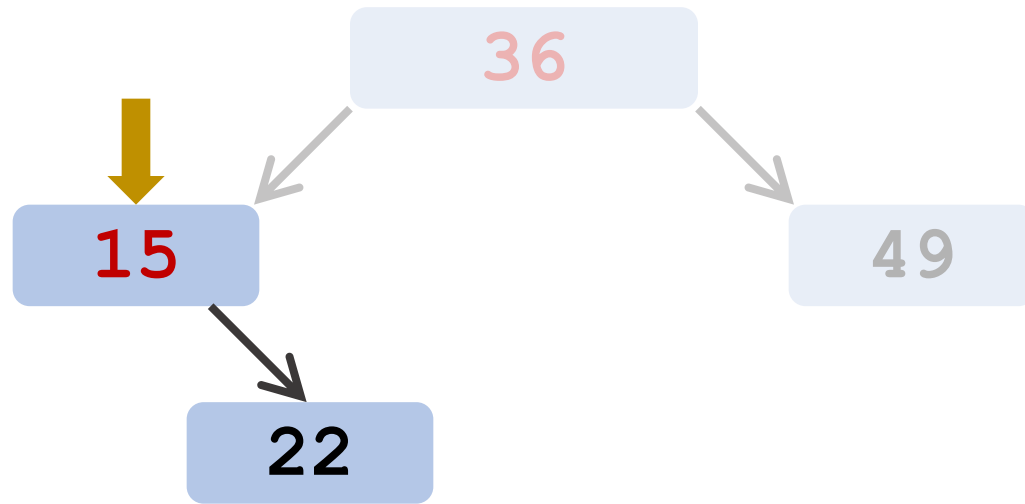
Insert “**key=31**”



1. Create new vertex **31**

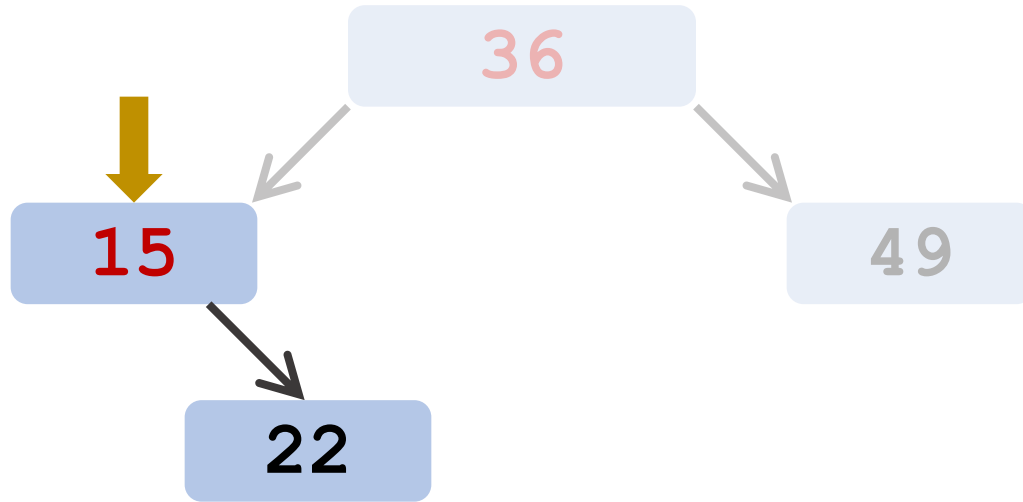
2. **key** < **36** → Go to left.

Insert “key=31”



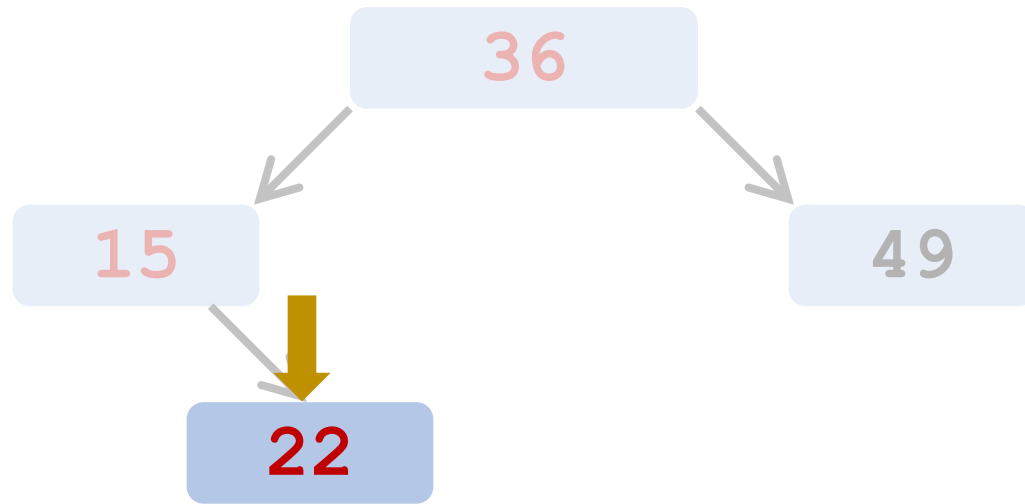
1. Create new vertex **31**
2. $\text{key} < 36 \Rightarrow$ Go to left.

Insert “key=31”



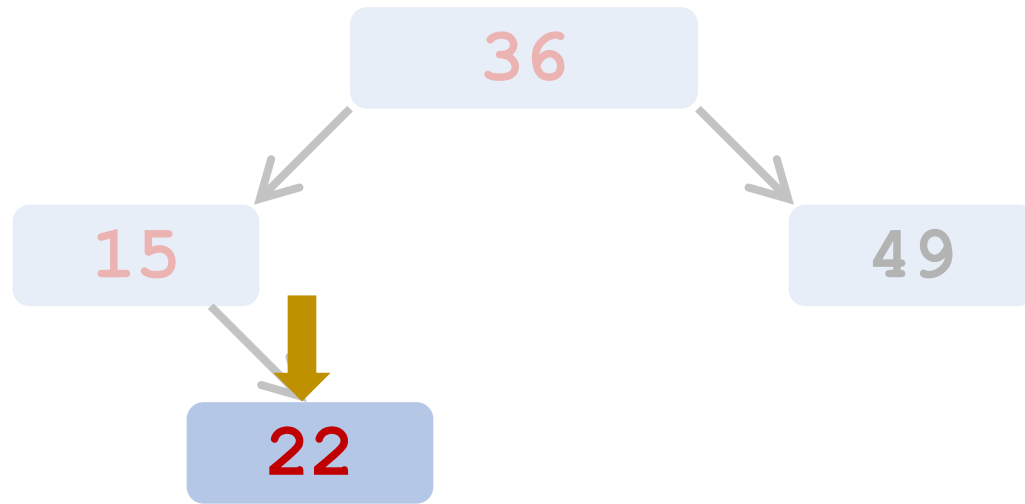
1. Create new vertex **31**
2. $\text{key} < 36 \Rightarrow$ Go to left.
3. $\text{key} > 15 \Rightarrow$ Go to right.

Insert “key=31”



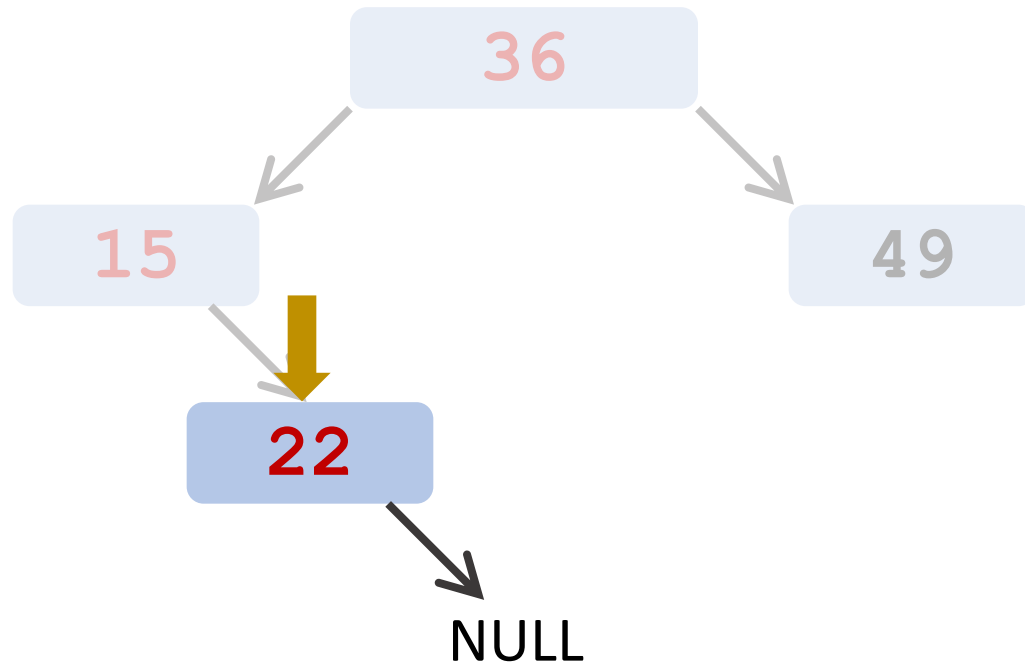
1. Create new vertex **31**
2. $\text{key} < 36 \Rightarrow$ Go to left.
3. $\text{key} > 15 \Rightarrow$ Go to right.

Insert “key=31”



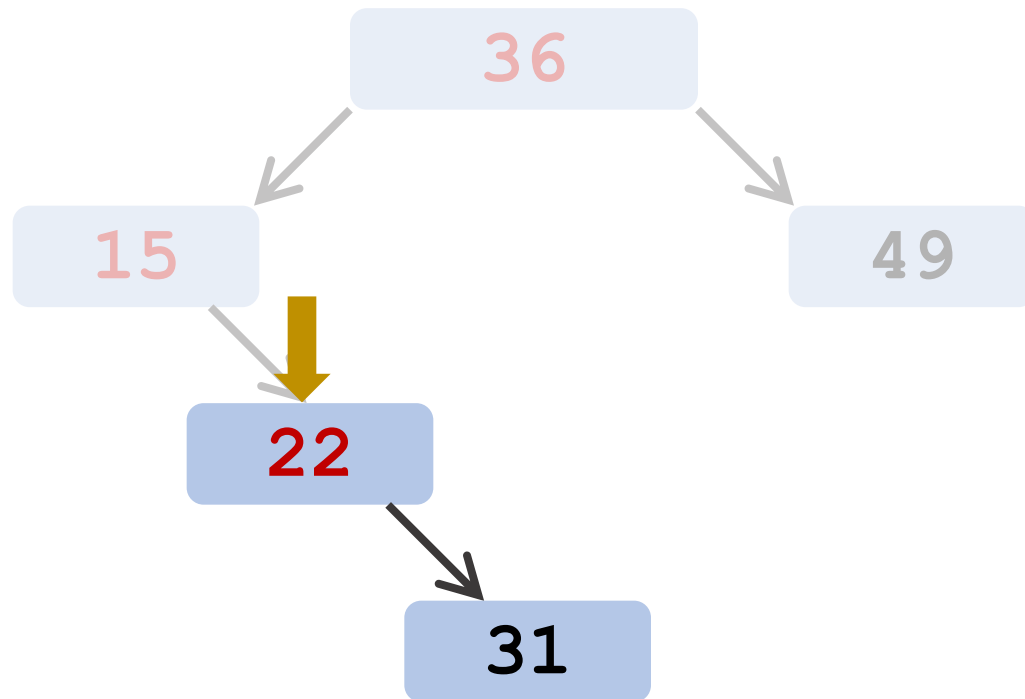
1. Create new vertex **31**
2. $\text{key} < 36 \rightarrow$ Go to left.
3. $\text{key} > 15 \rightarrow$ Go to right.
4. $\text{key} > 22 \rightarrow$ Go to right.

Insert “key=31”



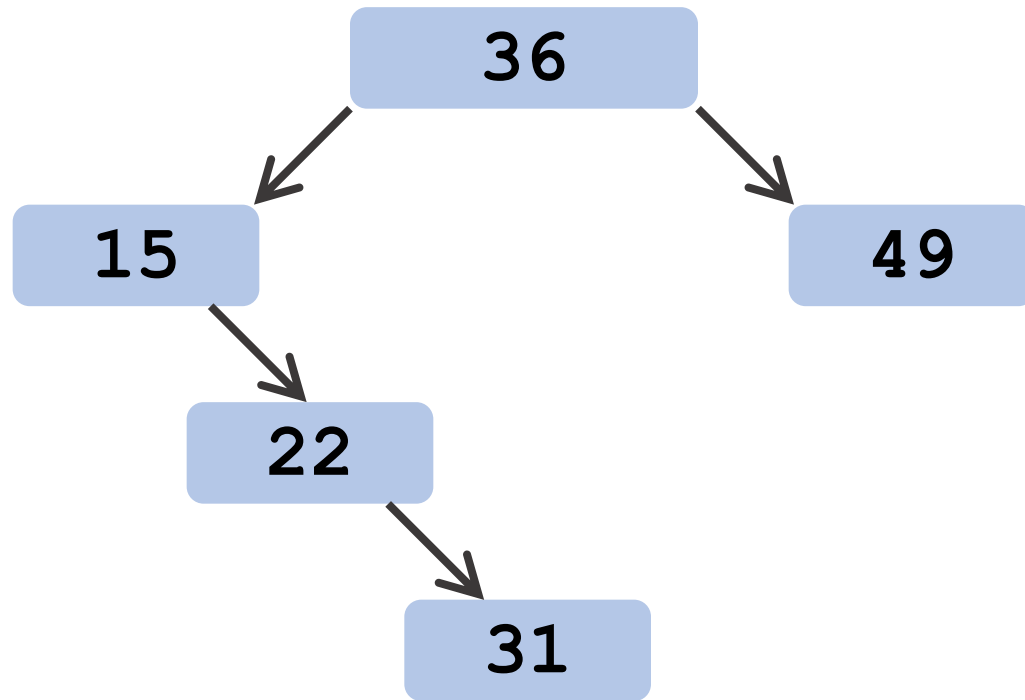
1. Create new vertex **31**
2. $\text{key} < 36 \rightarrow$ Go to left.
3. $\text{key} > 15 \rightarrow$ Go to right.
4. $\text{key} > 22 \rightarrow$ Go to right.
5. Right child is NULL

Insert “key=31”



1. Create new vertex **31**
2. $\text{key} < 36 \rightarrow$ Go to left.
3. $\text{key} > 15 \rightarrow$ Go to right.
4. $\text{key} > 22 \rightarrow$ Go to right.
5. Right child is NULL \rightarrow Make “31” the right child.

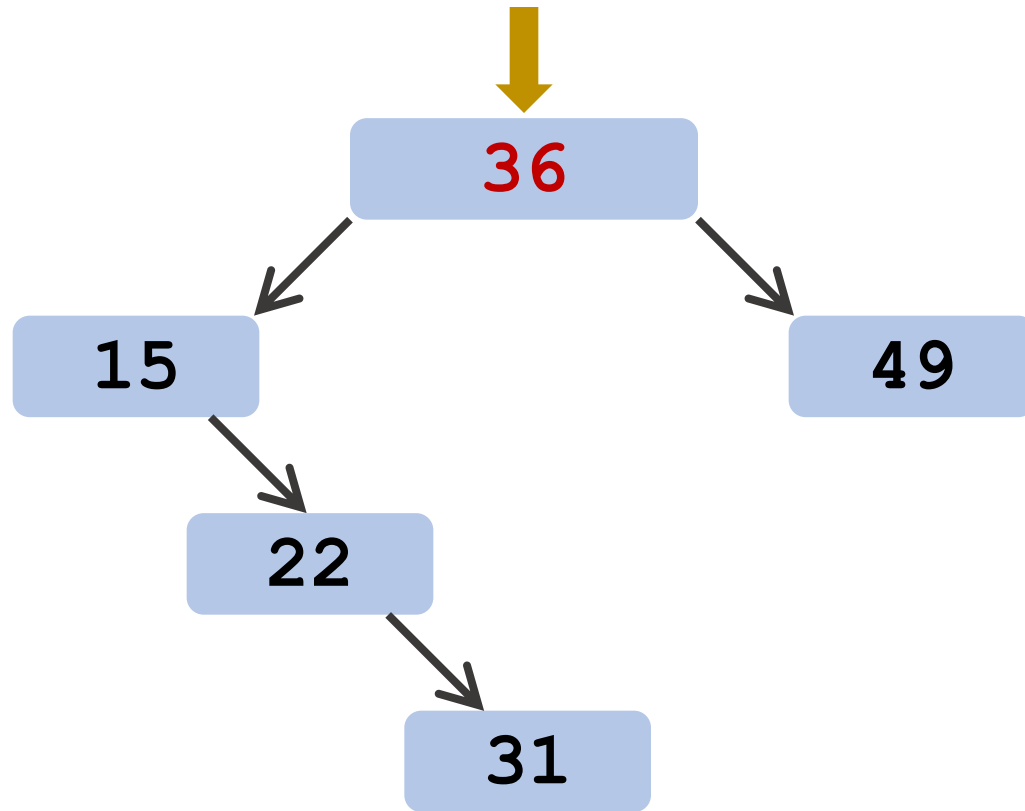
Insert “key=6”



1. Create new vertex

6

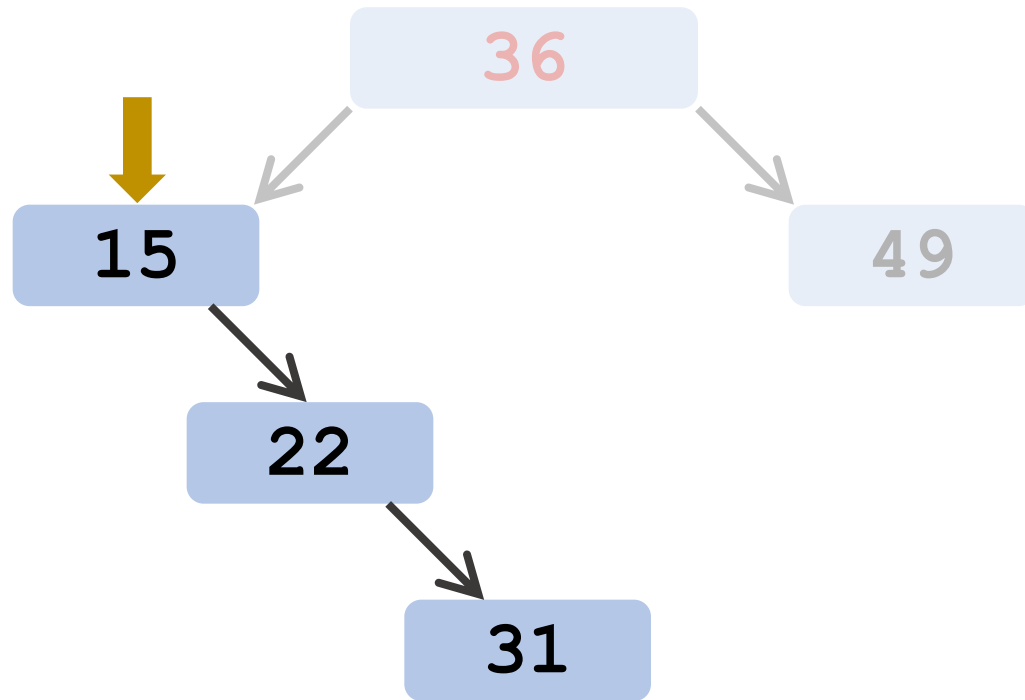
Insert “key=6”



1. Create new vertex 6

2. $\text{key} < 36 \Rightarrow$ Go to left.

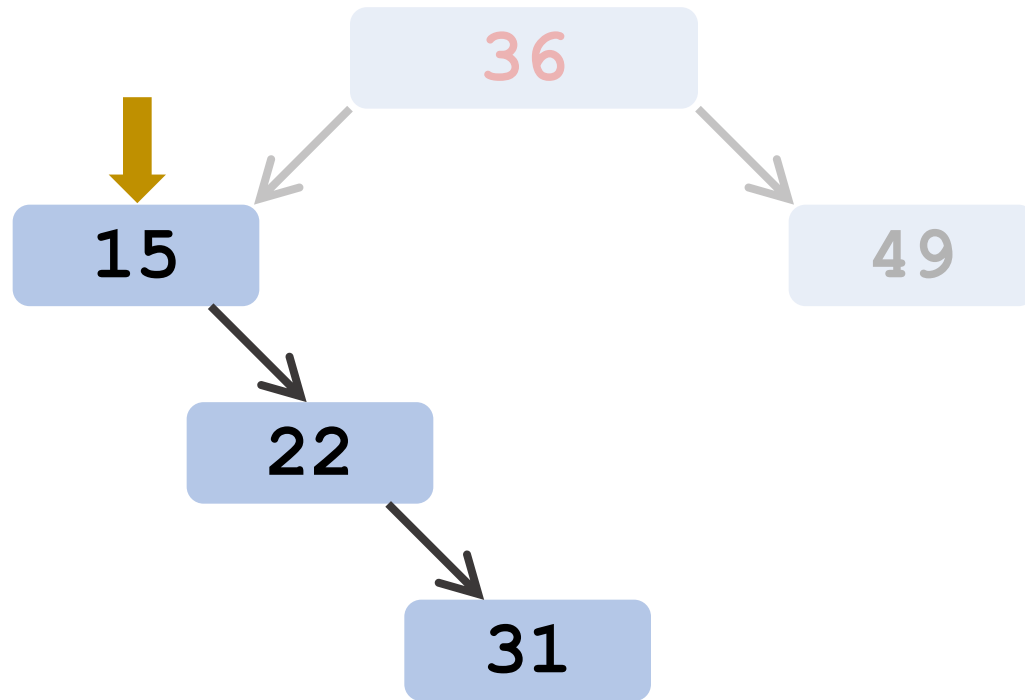
Insert “key=6”



1. Create new vertex 6

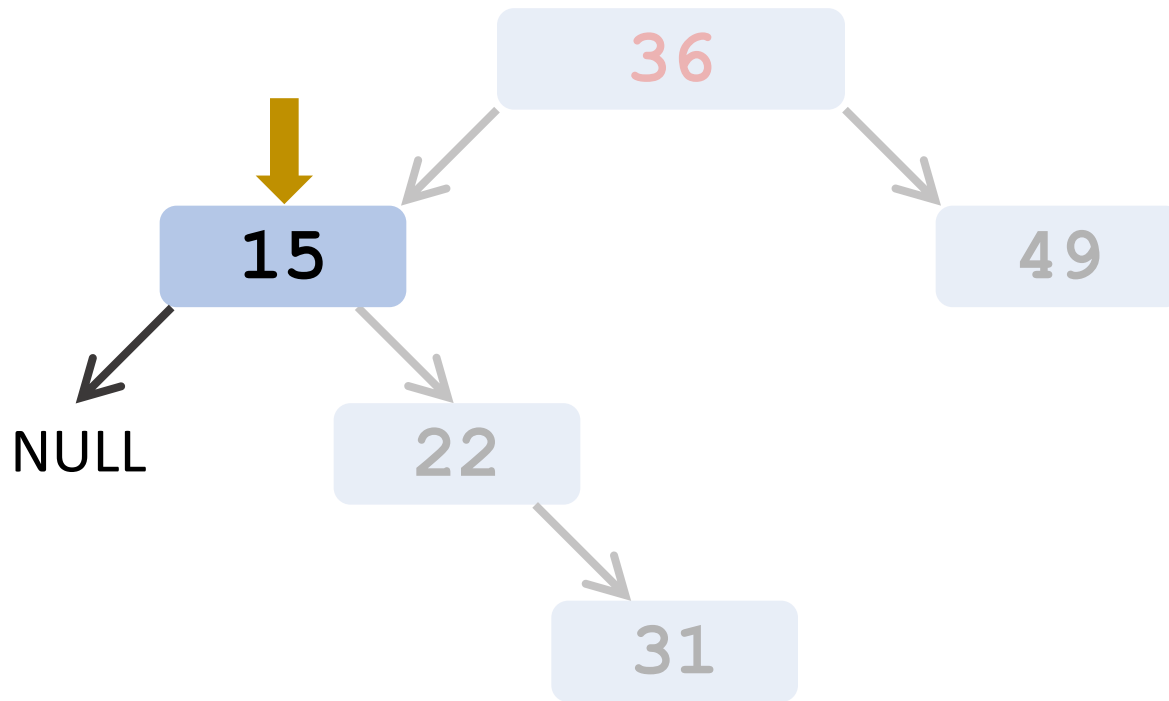
2. $\text{key} < 36 \Rightarrow$ Go to left.

Insert “key=6”



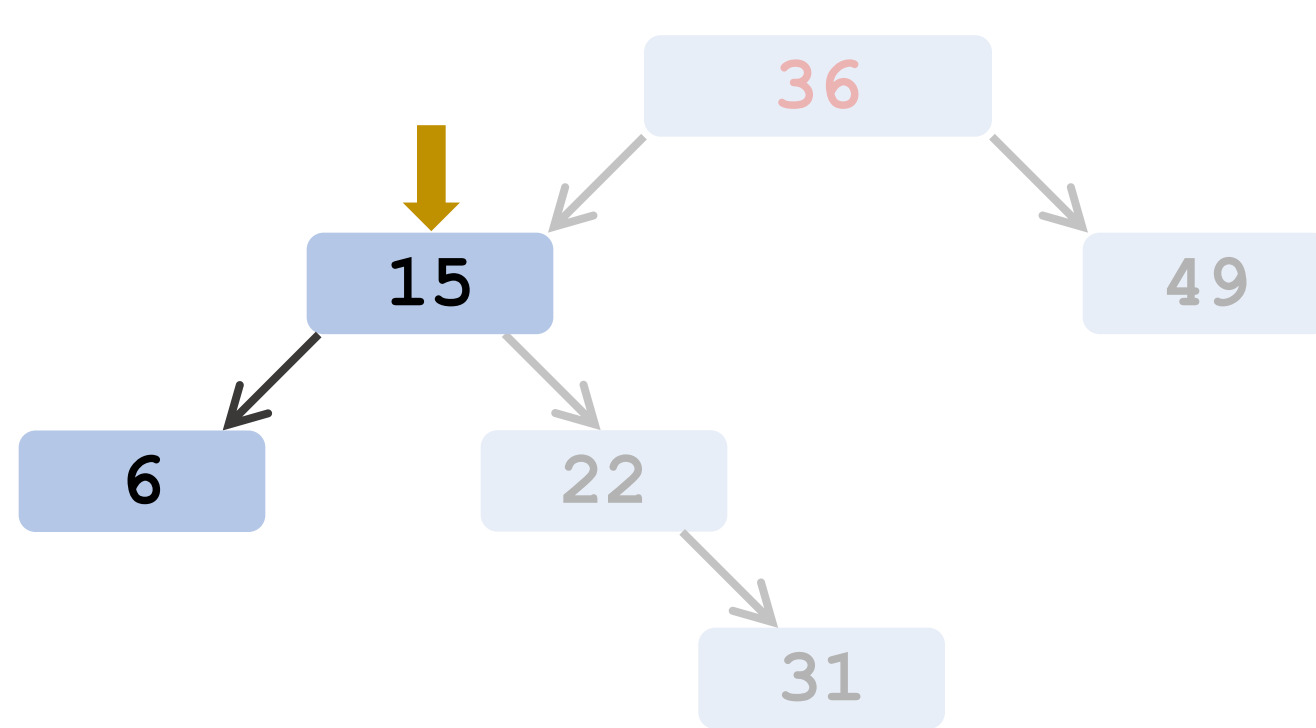
1. Create new vertex 6
2. $\text{key} < 36 \Rightarrow$ Go to left.
3. $\text{key} < 15 \Rightarrow$ Go to left.

Insert “key=6”




1. Create new vertex 6
2. $\text{key} < 36 \Rightarrow$ Go to left.
3. $\text{key} < 15 \Rightarrow$ Go to left.
4. Left child is NULL


Insert “key=6”





1. Create new vertex **6**
2. $\text{key} < 36 \rightarrow$ Go to left.
3. $\text{key} < 15 \rightarrow$ Go to left.
4. Left child is NULL \rightarrow Make “6” the left child.



```
struct vertex* insert(struct vertex* root, int key) {  
    if (root == NULL) { // the tree is empty  
        struct vertex* r = newVertex(key);  
        return r; // new root  
    }  
    else { // recur down the tree  
        if (key < root->key)  
            root->left = insert(root->left, key);  
        else if (key > root->key)  
            root->right = insert(root->right, key);  
        return root; // the root is unchanged  
    }  
}
```

```
struct vertex* insert(struct vertex* root, int key) {  
     if (root == NULL) { // the tree is empty  
        struct vertex* r = newVertex(key);  
        return r; // new root  
    }  
    else { // recur down the tree  
        if (key < root->key)  
            root->left = insert(root->left, key);  
        else if (key > root->key)  
            root->right = insert(root->right, key);  
        return root; // the root is unchanged  
    }  
}
```

```
struct vertex* insert(struct vertex* root, int key) {  
    if (root == NULL) { // the tree is empty  
        struct vertex* r = newVertex(key);  
        return r; // new root  
    }  
     else { // recur down the tree  
        if (key < root->key)  
            root->left = insert(root->left, key);  
        else if (key > root->key)  
            root->right = insert(root->right, key);  
        return root; // the root is unchanged  
    }  
}
```

```
struct vertex* insert(struct vertex* root, int key) {  
    if (root == NULL) { // the tree is empty  
        struct vertex* r = newVertex(key);  
        return r; // new root  
    }  
    else { // recur down the tree  
         if (key < root->key)  
            root->left = insert(root->left, key);  
        else if (key > root->key)  
            root->right = insert(root->right, key);  
        return root; // the root is unchanged  
    }  
}
```

```
struct vertex* insert(struct vertex* root, int key) {  
    if (root == NULL) { // the tree is empty  
        struct vertex* r = newVertex(key);  
        return r; // new root  
    }  
    else { // recur down the tree  
        if (key < root->key)  
            root->left = insert(root->left, key);  
         else if (key > root->key)  
            root->right = insert(root->right, key);  
        return root; // the root is unchanged  
    }  
}
```

```
struct vertex* insert(struct vertex* root, int key) {  
    if (root == NULL) { // the tree is empty  
        struct vertex* r = newVertex(key);  
        return r; // new root  
    }  
    else { // recur down the tree  
        if (key < root->key)  
            root->left = insert(root->left, key);  
        else if (key > root->key)  
            root->right = insert(root->right, key);  
         return root; // the root is unchanged  
    }  
}
```

Questions

Question

- Initially, the binary search tree is empty.
- The following keys are inserted sequentially:

19, 89, 64, 8, 9, 6, 4, 66, 76.

- Draw the tree after all the insertions.

Thank You!

<http://wangshusen.github.io/>