LECTURE 22

SET AND DEFAULTDICT

MCS 275 Spring 2021

Emily Dumas

LECTURE 22: SET AND DEFAULTDICT

Course bulletins:

- I hope to have Project 3 ready by Monday. It is due March 19.
- Thursday discussion students: Please attempt problem 1 of Worksheet 8 before discussion.

PLAN

- Wrap up trees unit
- Start language features unit

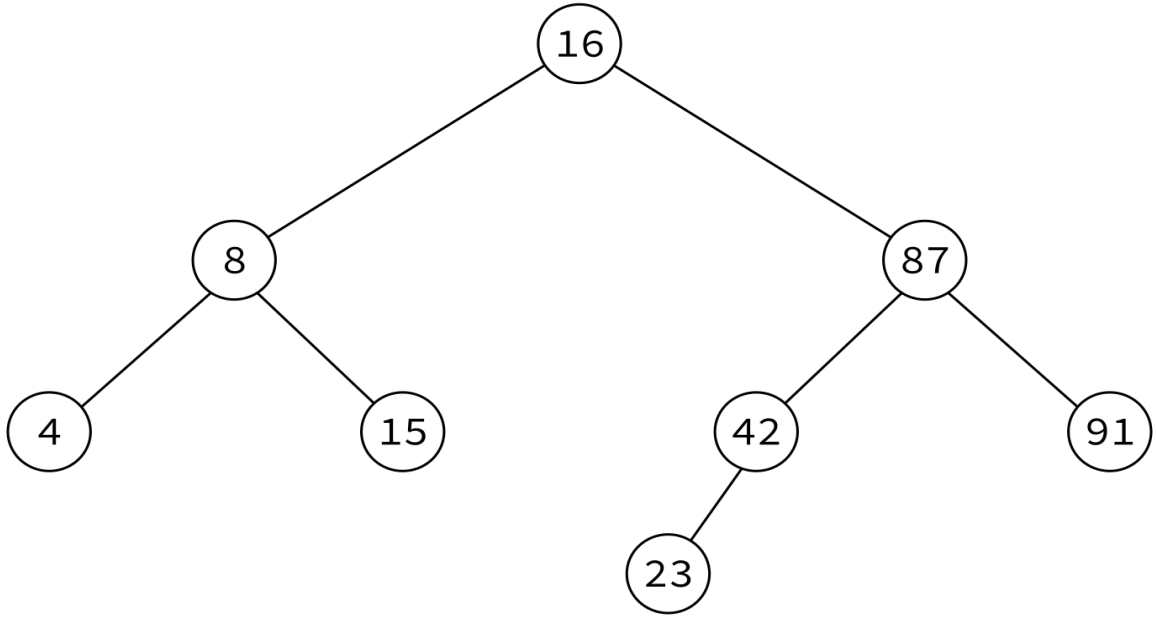
NAMED TRAVERSALS

The three most-often used recursive traversals:

- **preorder** - Node, left, right. Used to copy BSTs.
- **postorder** - Left, right, node. Used to delete BSTs.
- **inorder** - Left, node, right. Used to turn BST into sorted list.

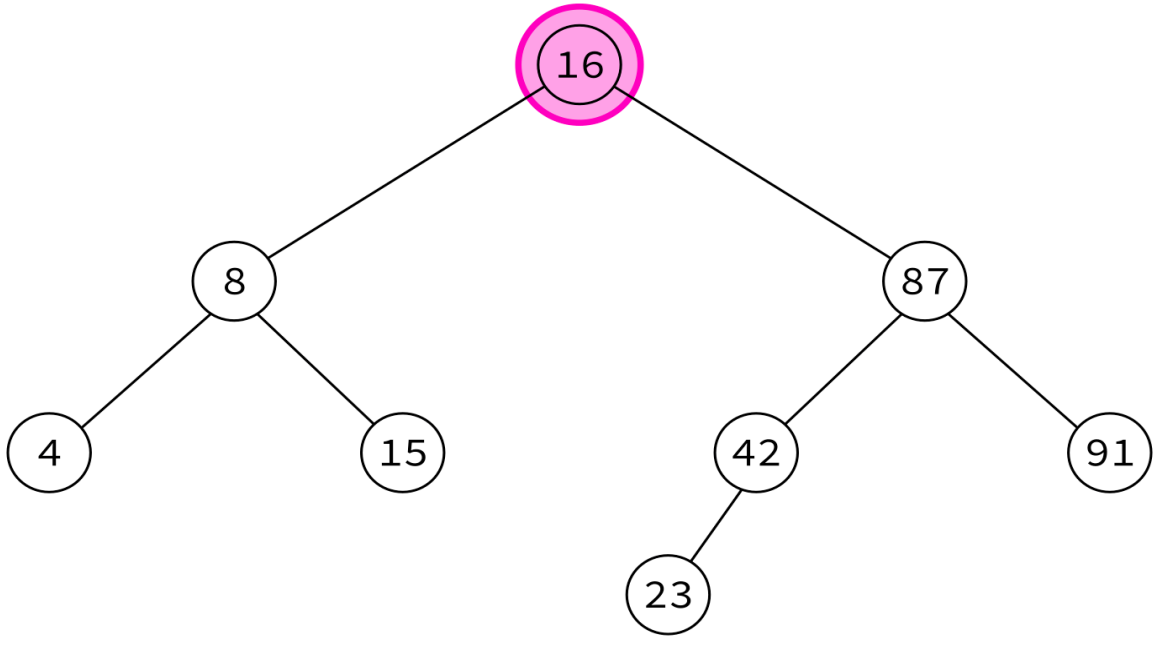
Note: They all visit left child before right child.

PREORDER TRAVERSAL



node, left, right

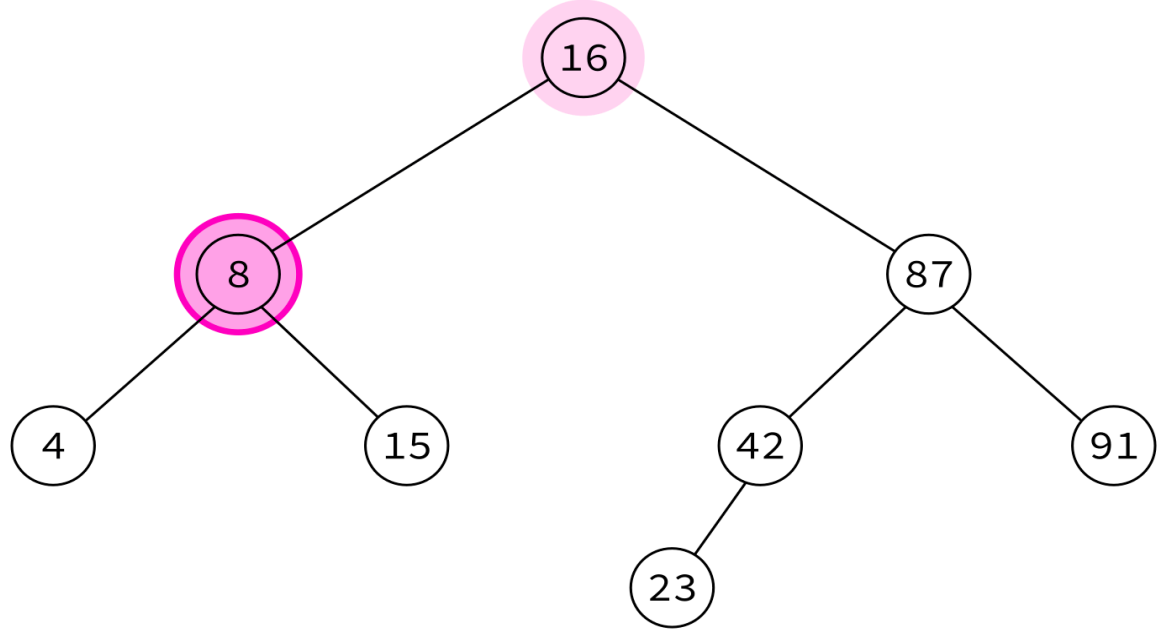
PREORDER TRAVERSAL




```
16,
```

node, left, right

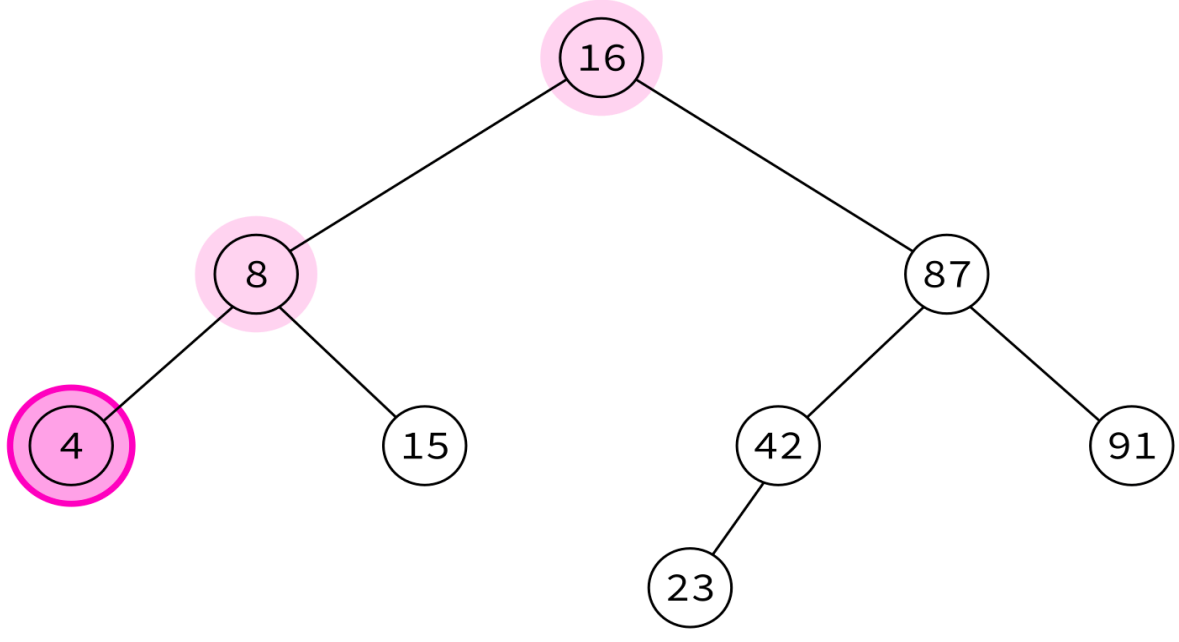
PREORDER TRAVERSAL




```
16, 8,
```

node, left, right

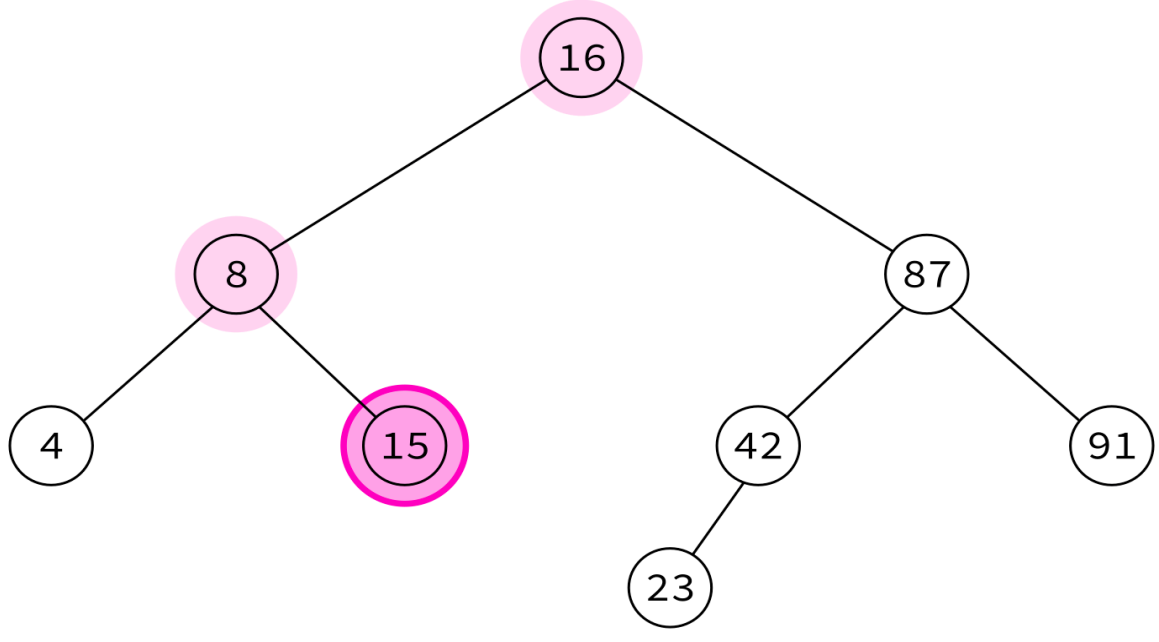
PREORDER TRAVERSAL




```
16, 8, 4,
```

node, left, right

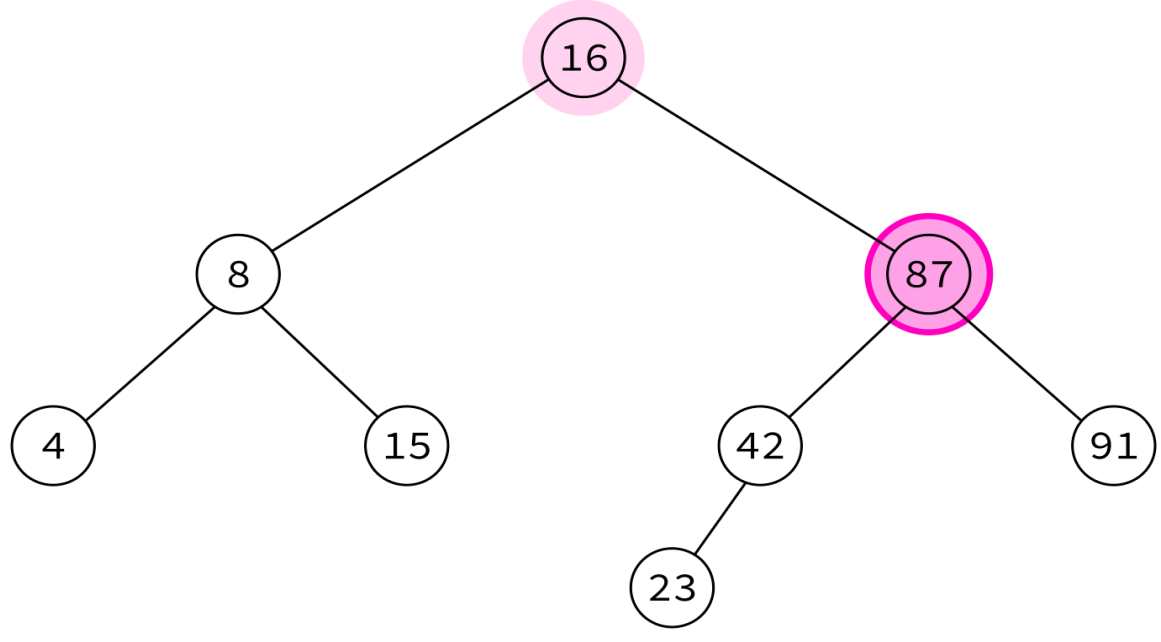
PREORDER TRAVERSAL




```
16, 8, 4, 15,
```

node, left, right

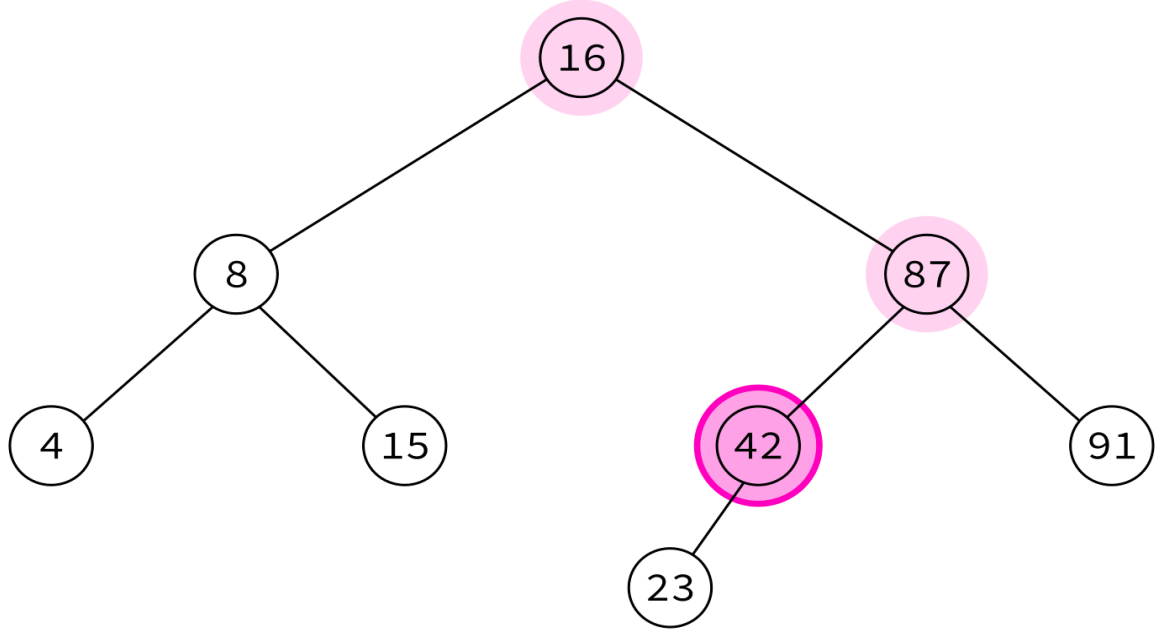
PREORDER TRAVERSAL




```
16, 8, 4, 15, 87,
```

node, left, right

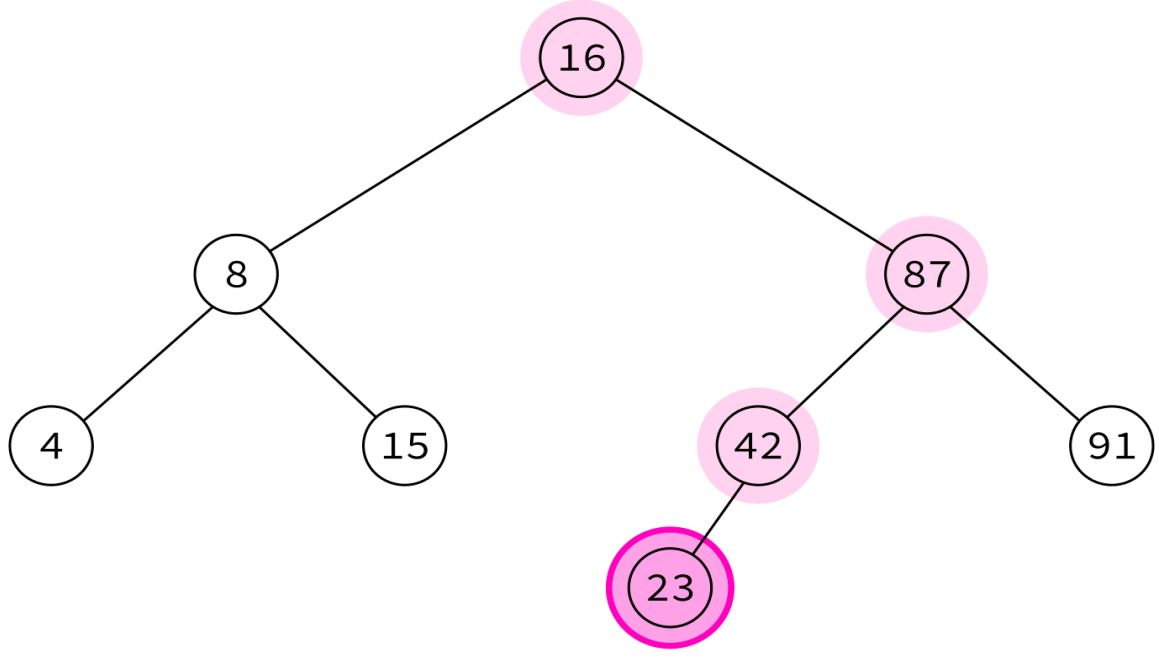
PREORDER TRAVERSAL




```
16, 8, 4, 15, 87, 42,
```

node, left, right

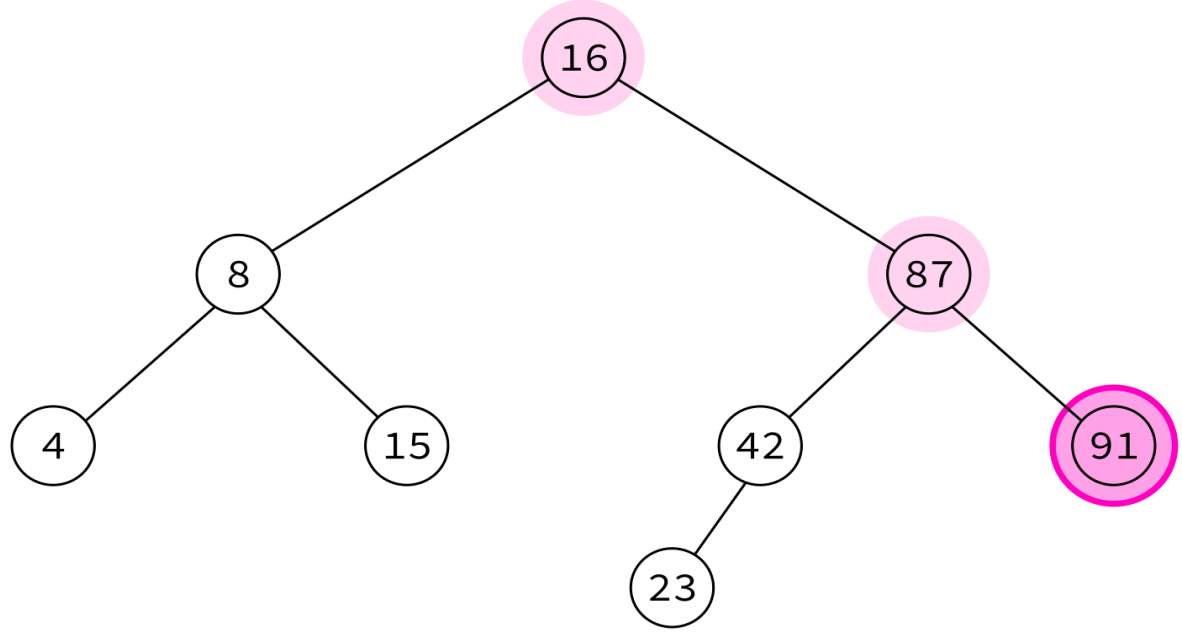
PREORDER TRAVERSAL




```
16, 8, 4, 15, 87, 42, 23,
```

node, left, right

PREORDER TRAVERSAL



`16, 8, 4, 15, 87, 42, 23, 91`

node, left, right

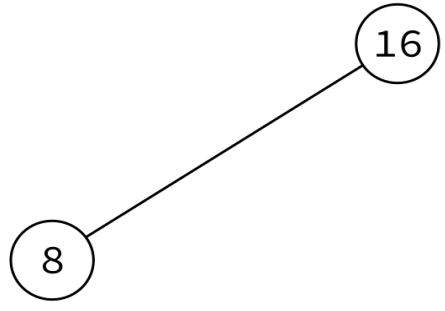
PREORDER TRAVERSAL

Typical use: Make a copy of the tree.

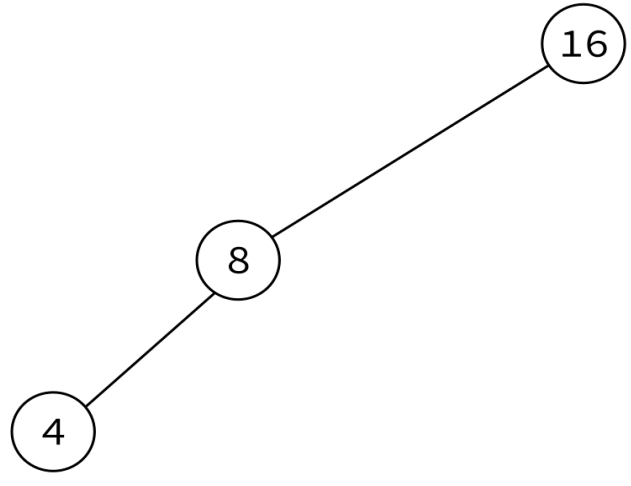
Insert the keys into an empty BST in this order to recreate the original tree.

16, 8, 4, 15, 87, 42, 23, 91

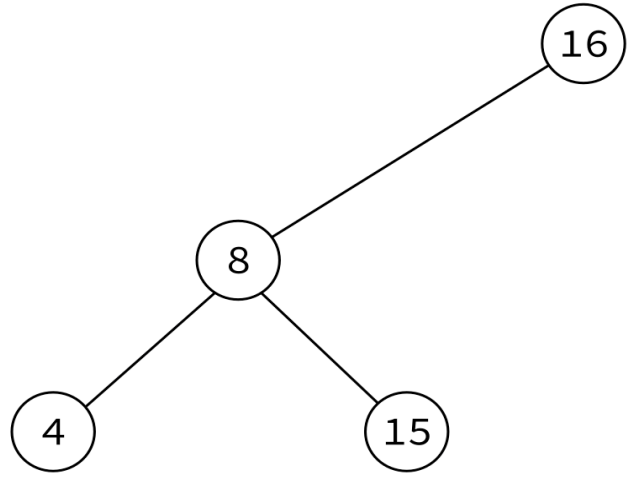
16, 8, 4, 15, 87, 42, 23, 91



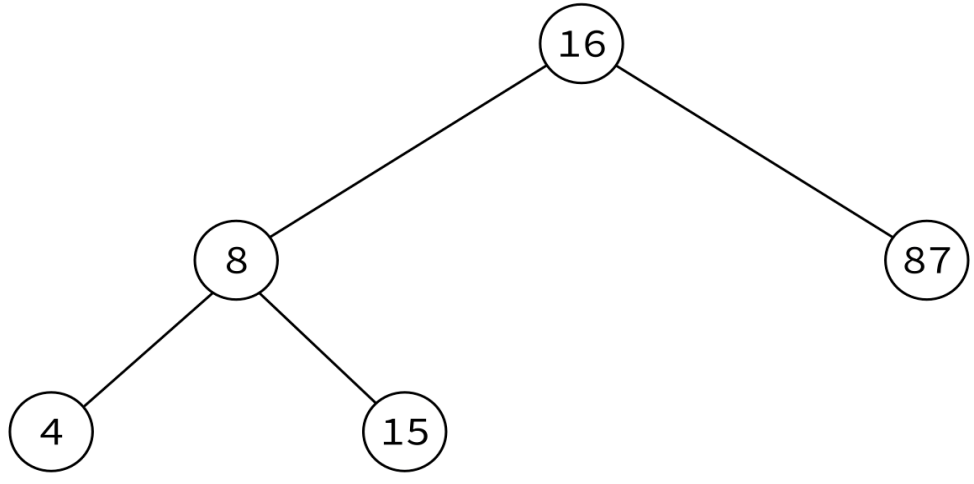
16, 8, 4, 15, 87, 42, 23, 91



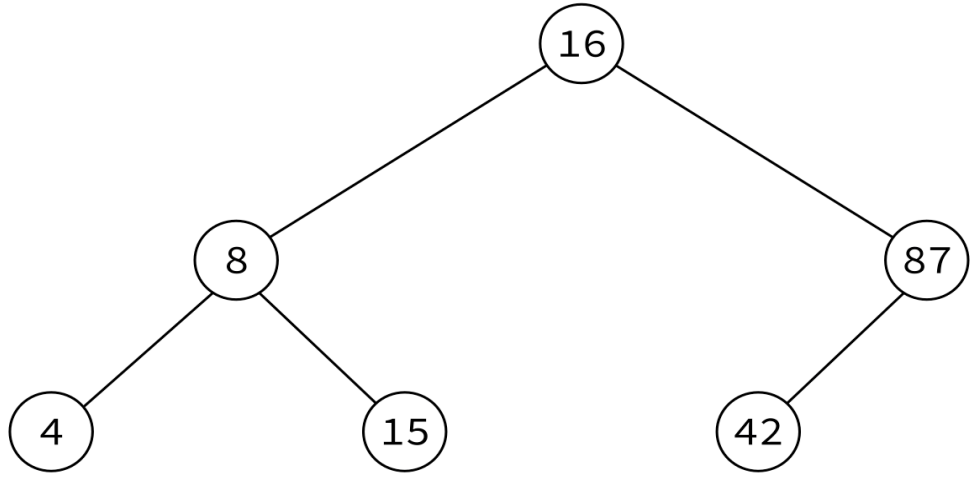
16, 8, 4, 15, 87, 42, 23, 91



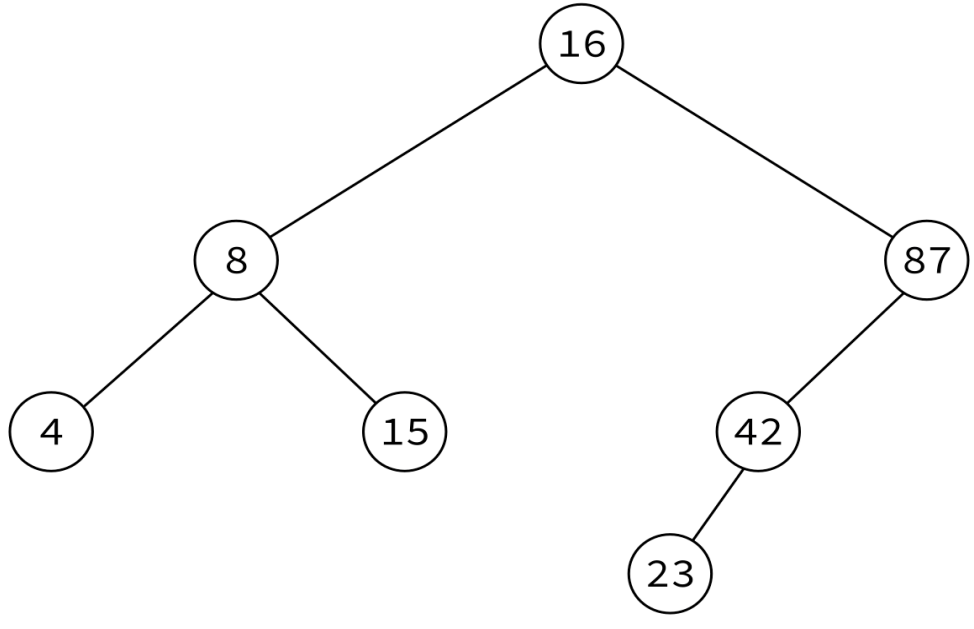
16, 8, 4, 15, 87, 42, 23, 91



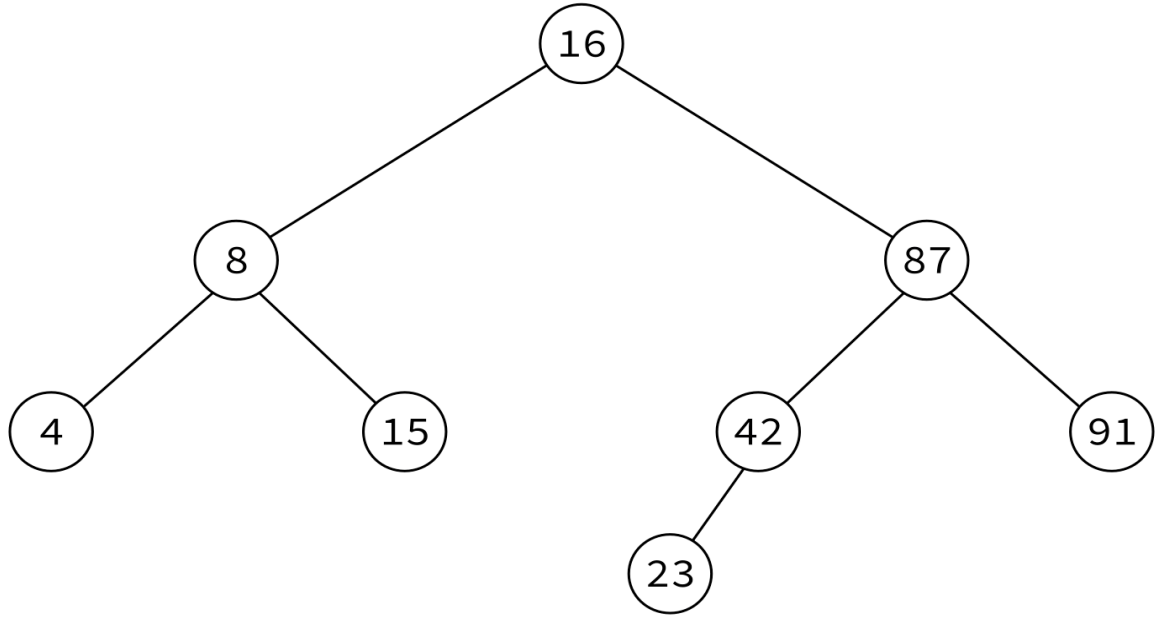
16, 8, 4, 15, 87, 42, 23, 91



16, 8, 4, 15, 87, 42, 23, 91

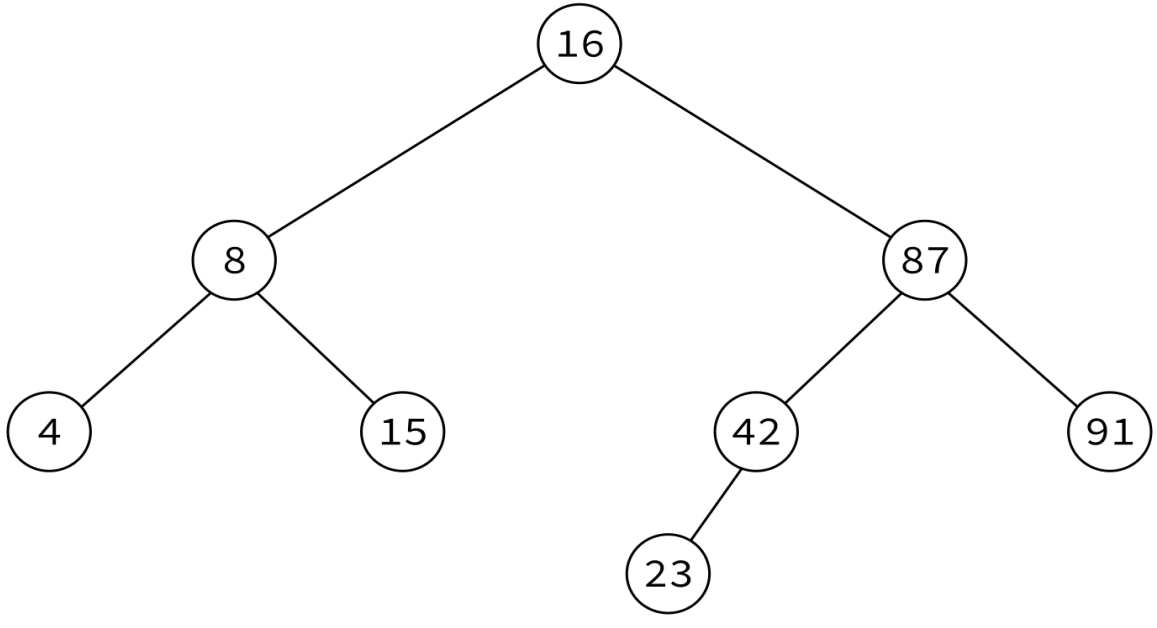


16, 8, 4, 15, 87, 42, 23, 91



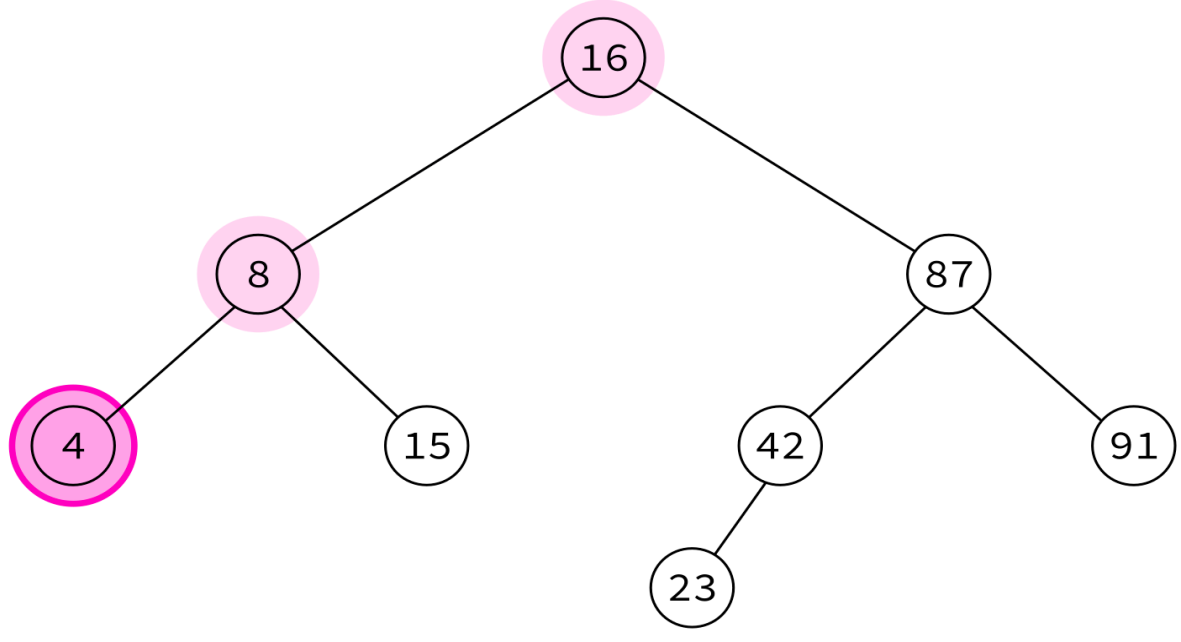
16, 8, 4, 15, 87, 42, 23, 91

POSTORDER TRAVERSAL



left, right, **node**

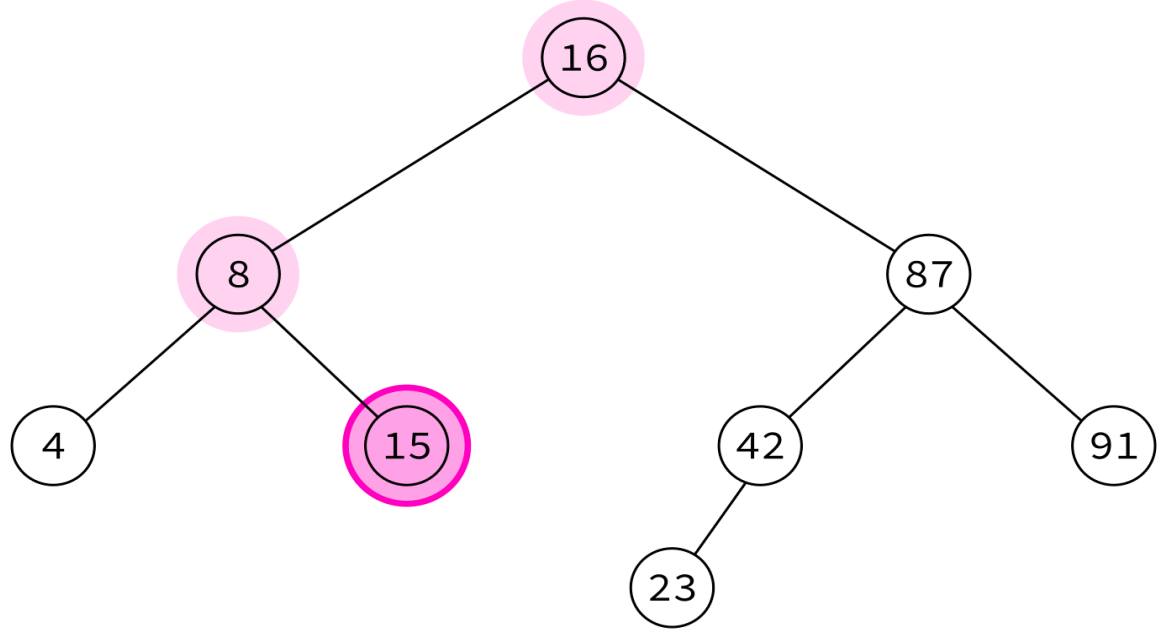
POSTORDER TRAVERSAL



4,

left, right, **node**

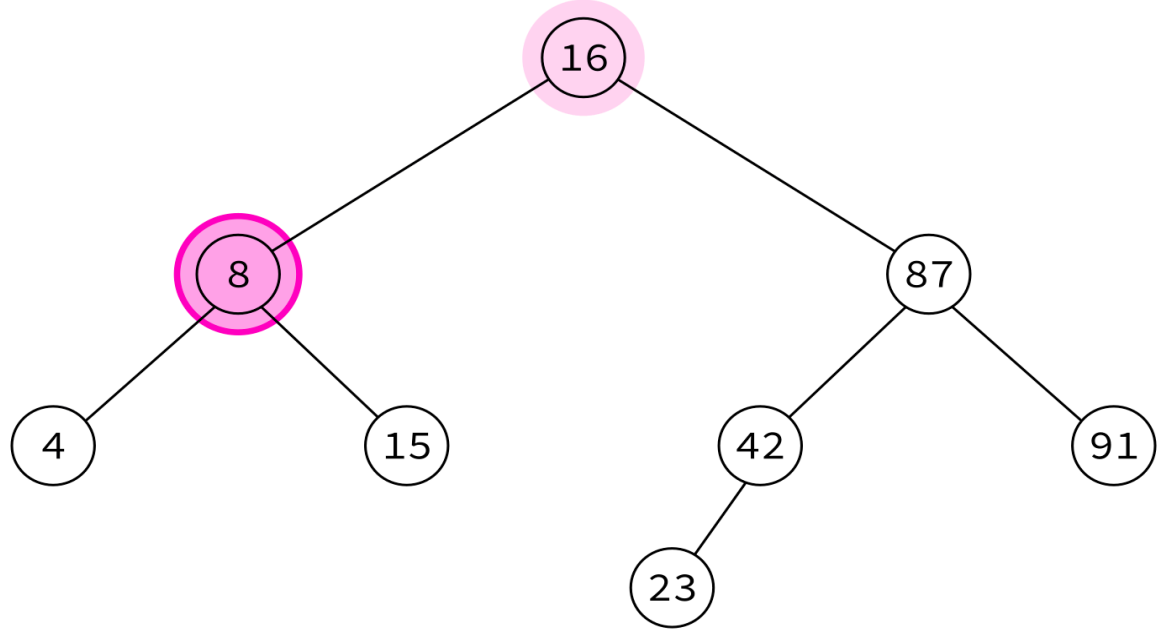
POSTORDER TRAVERSAL




```
4, 15,
```

left, right, **node**

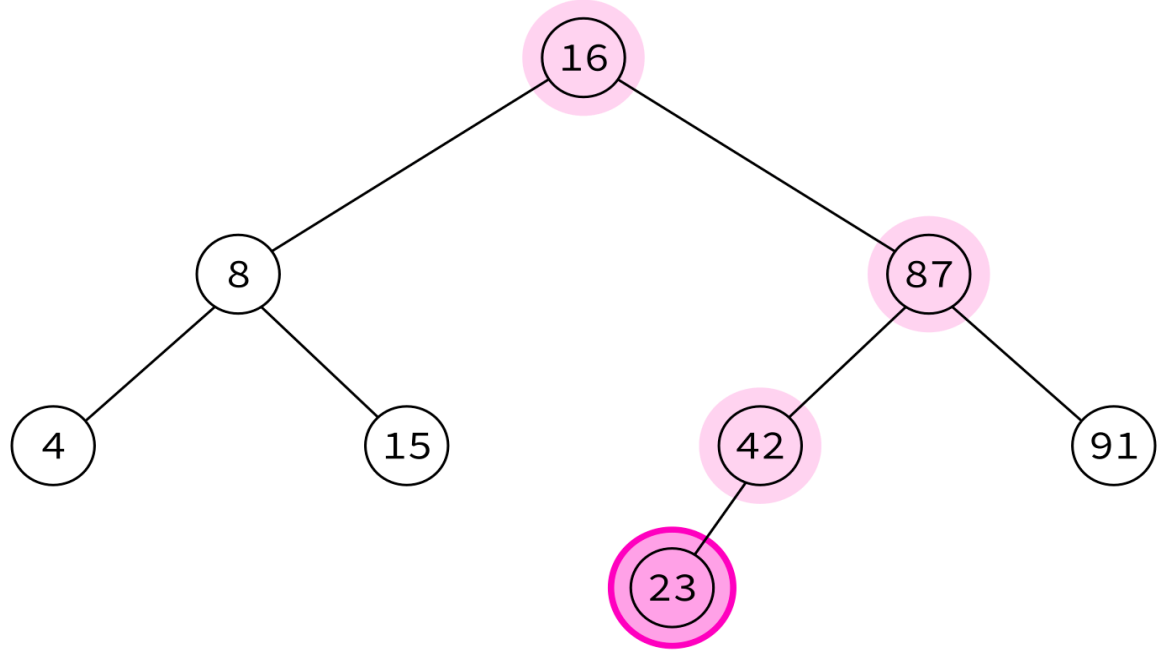
POSTORDER TRAVERSAL




```
4, 15, 8,
```

left, right, **node**

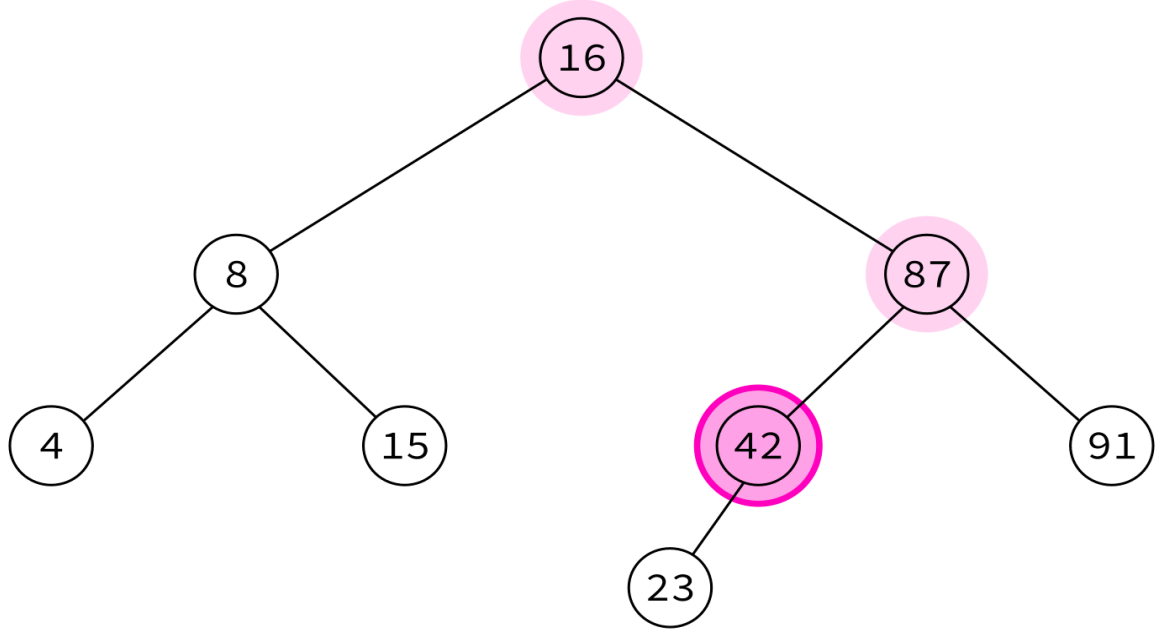
POSTORDER TRAVERSAL




```
4, 15, 8, 23,
```

left, right, **node**

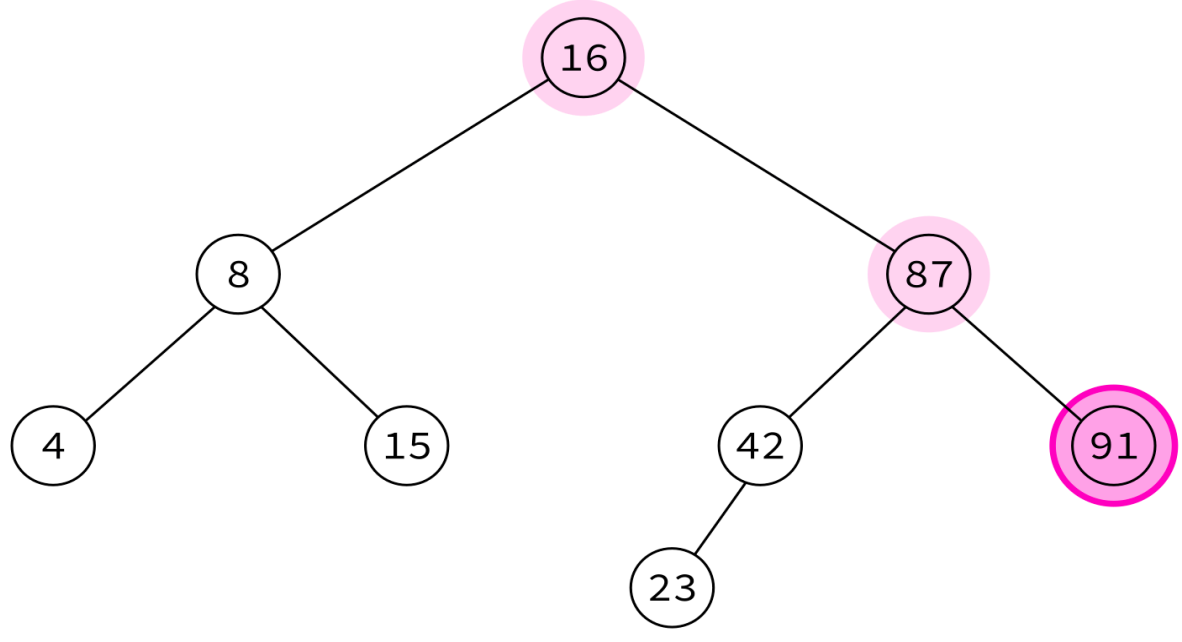
POSTORDER TRAVERSAL




```
4, 15, 8, 23, 42,
```

left, right, **node**

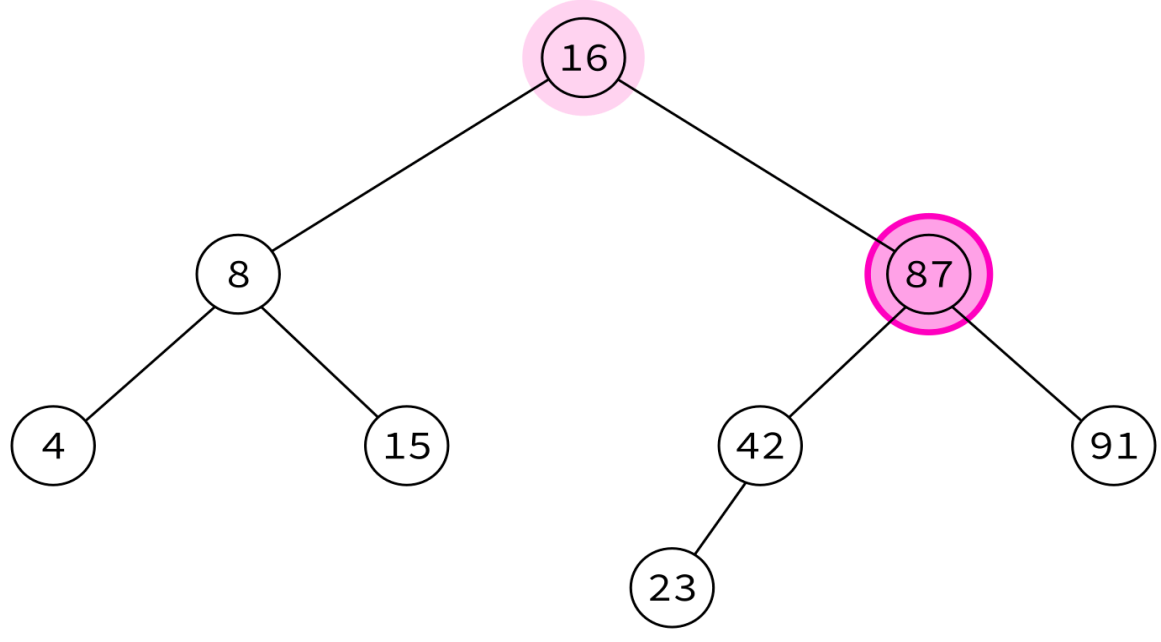
POSTORDER TRAVERSAL




```
4, 15, 8, 23, 42, 91,
```

left, right, **node**

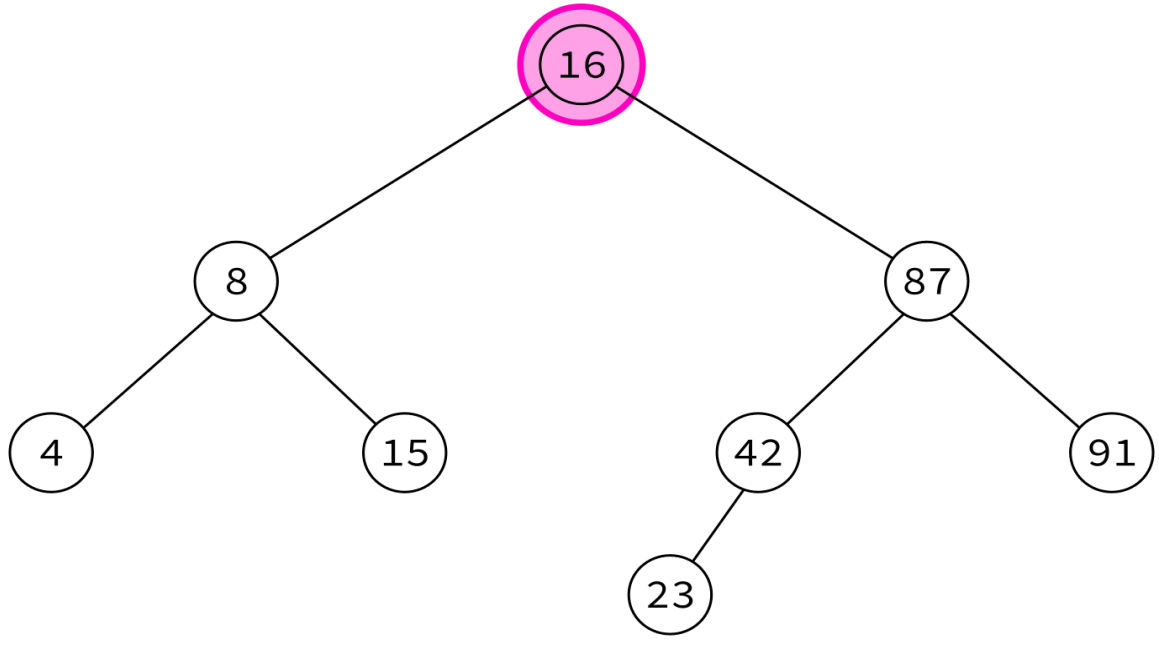
POSTORDER TRAVERSAL




```
4, 15, 8, 23, 42, 91, 87,
```

left, right, **node**

POSTORDER TRAVERSAL



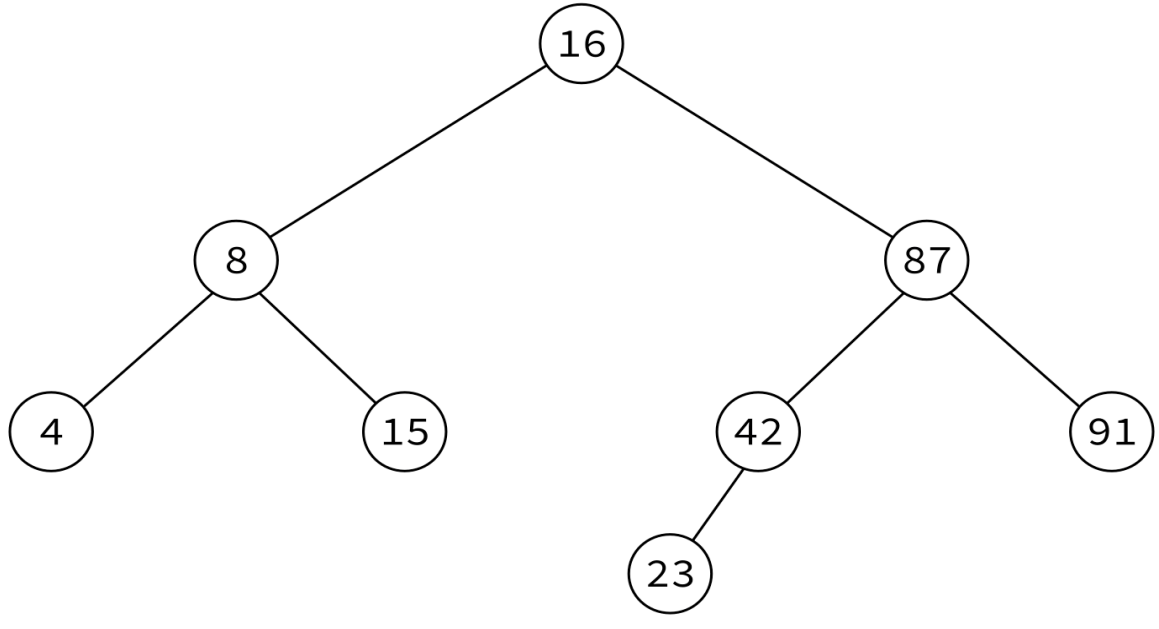
`4, 15, 8, 23, 42, 91, 87, 16`

left, right, **node**

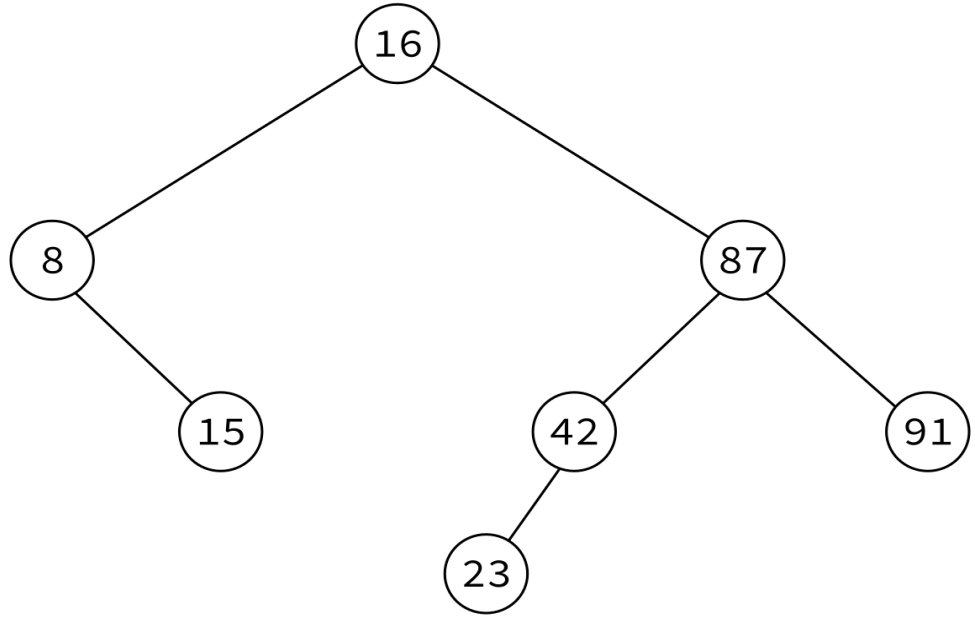
POSTORDER TRAVERSAL

Typical use: Delete the tree.

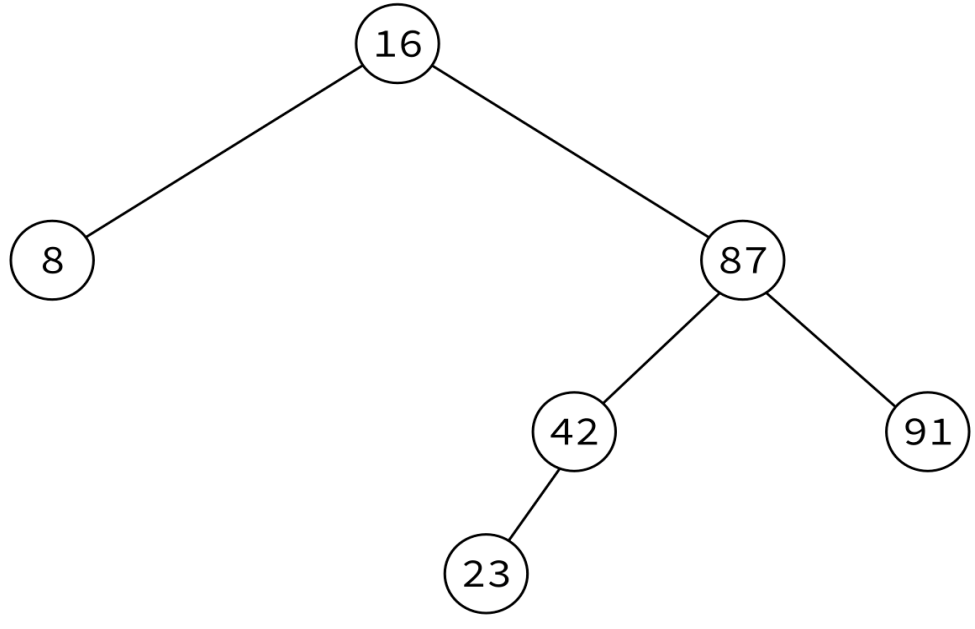
If you delete keys in postorder, then you will only ever be removing nodes without children.



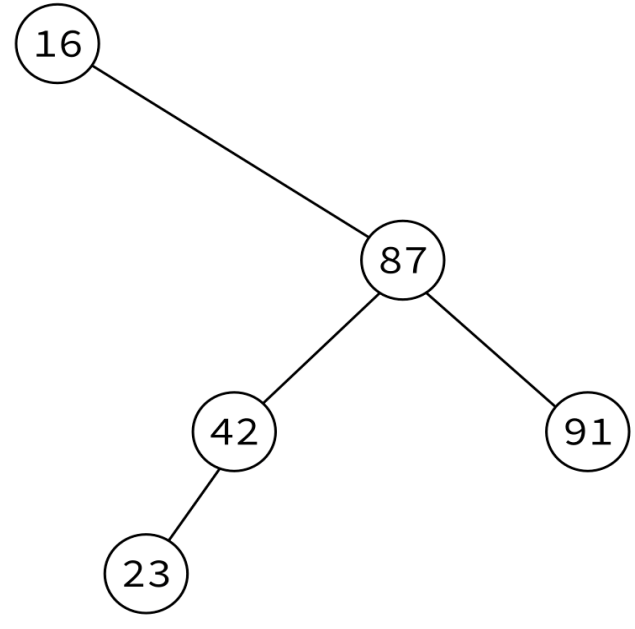
4, 15, 8, 23, 42, 91, 87, 16



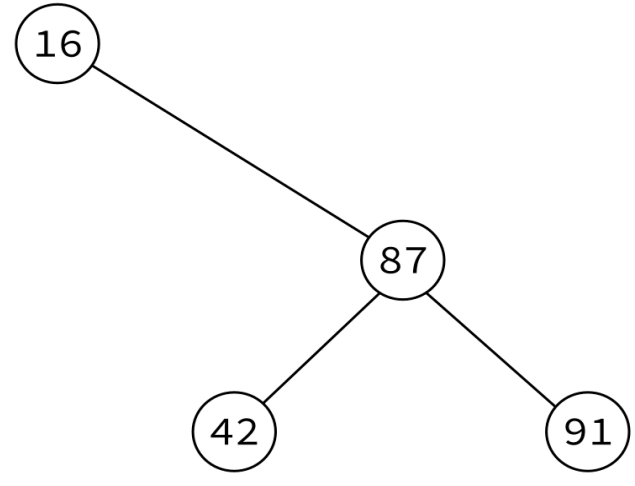
4, 15, 8, 23, 42, 91, 87, 16



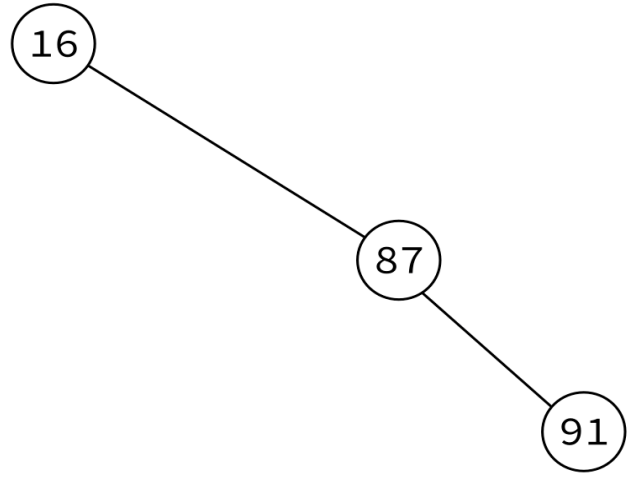
4, 15, 8, 23, 42, 91, 87, 16



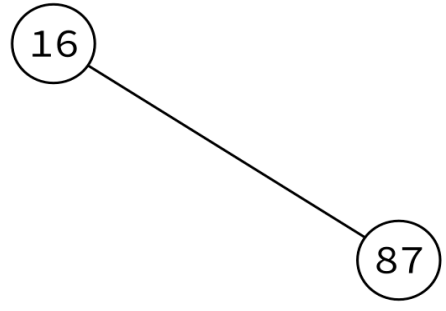
4, 15, 8, 23, 42, 91, 87, 16



4, 15, 8, 23, 42, 91, 87, 16



4, 15, 8, 23, 42, 91, 87, 16

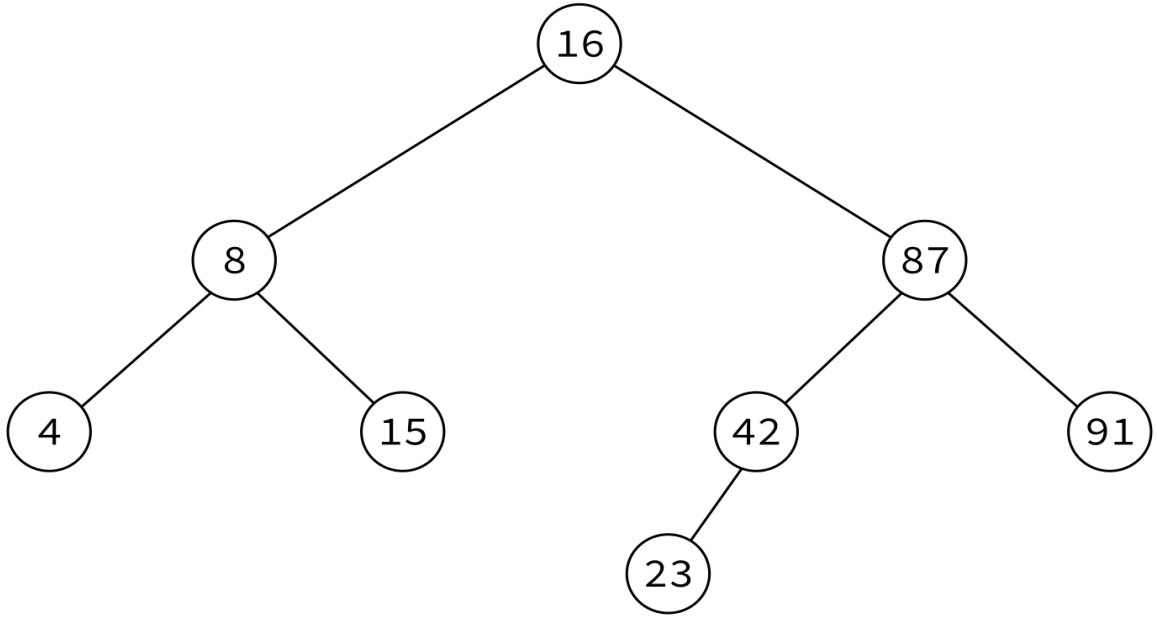


4, 15, 8, 23, 42, 91, 87, 16

4, 15, 8, 23, 42, 91, 87, 16

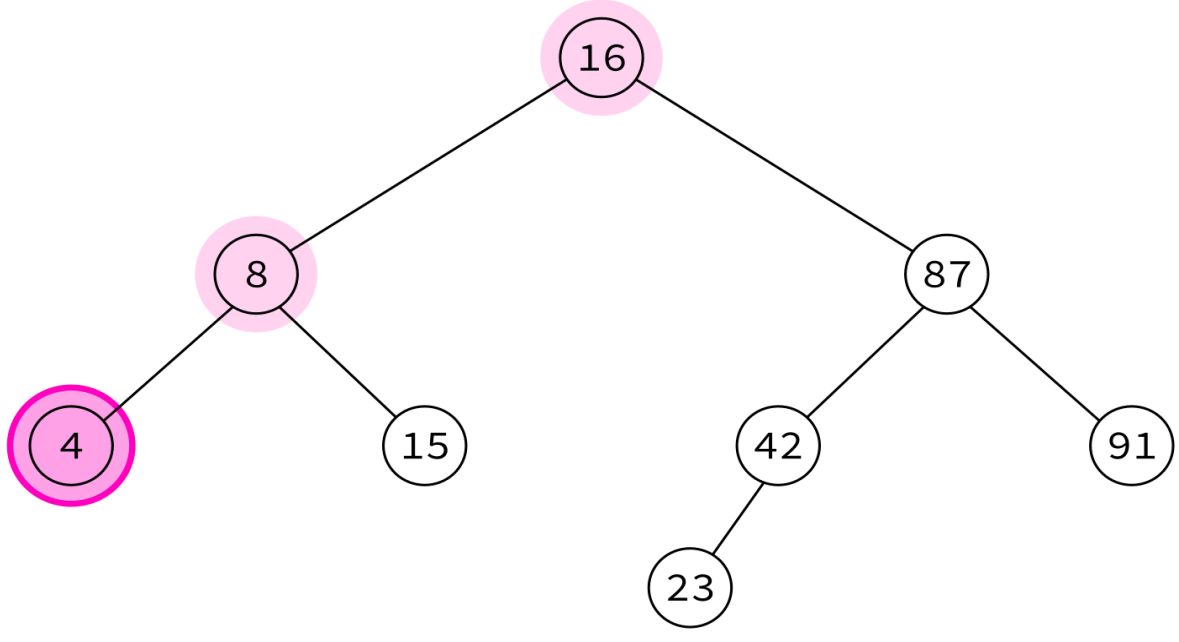
4, 15, 8, 23, 42, 91, 87, 16

INORDER TRAVERSAL



left, **node**, right

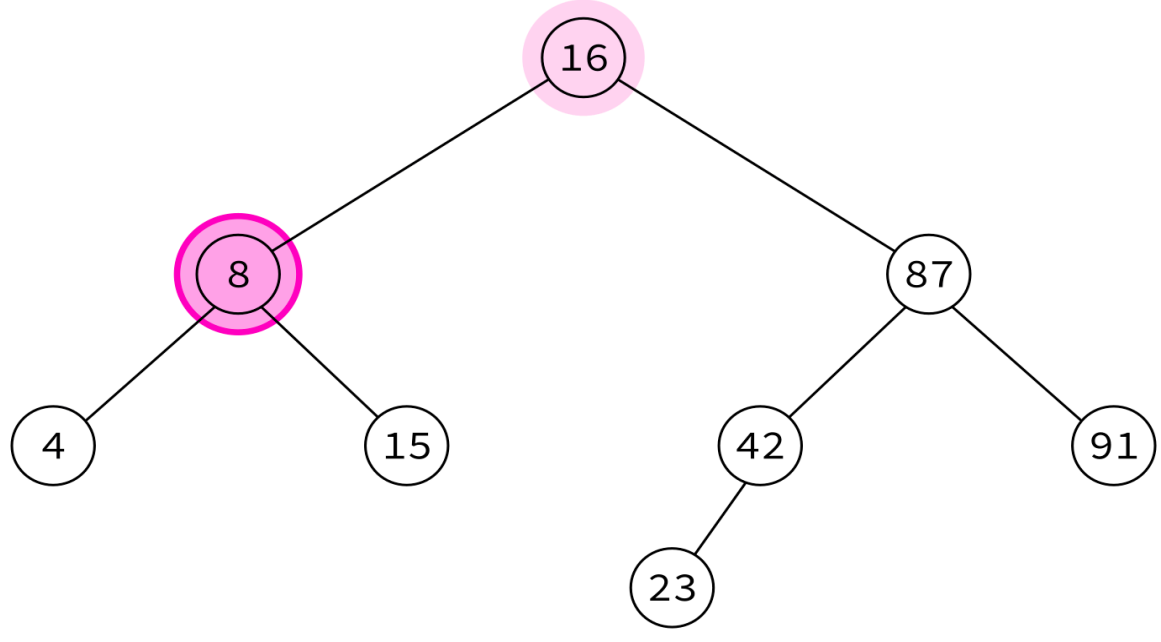
INORDER TRAVERSAL



4,

left, **node**, right

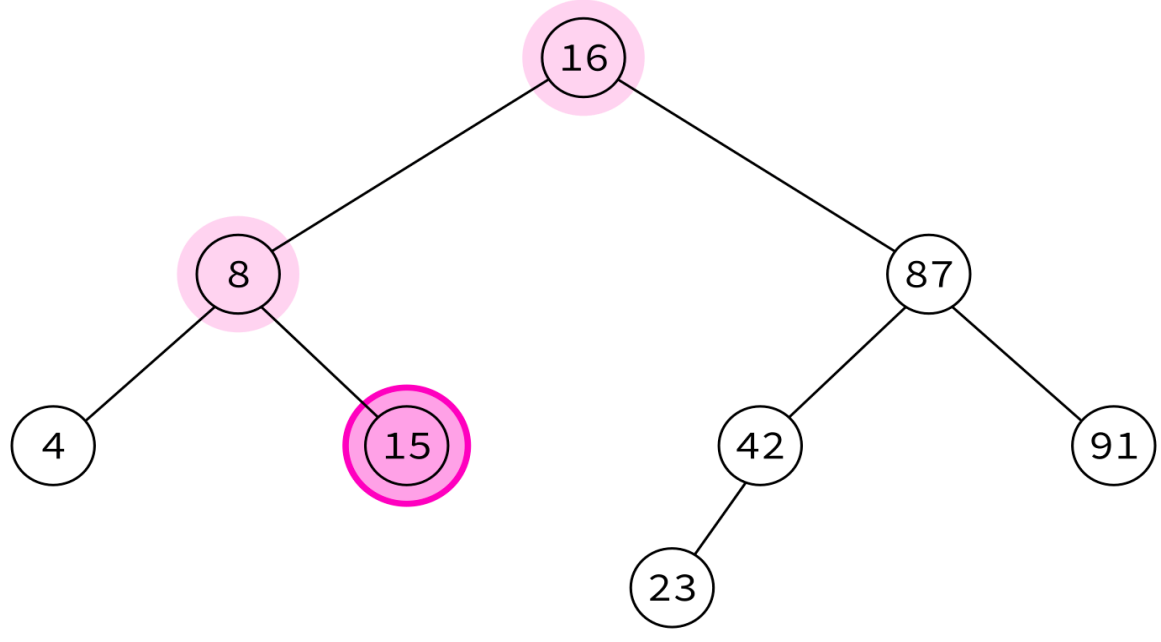
INORDER TRAVERSAL




```
4, 8,
```

left, **node**, right

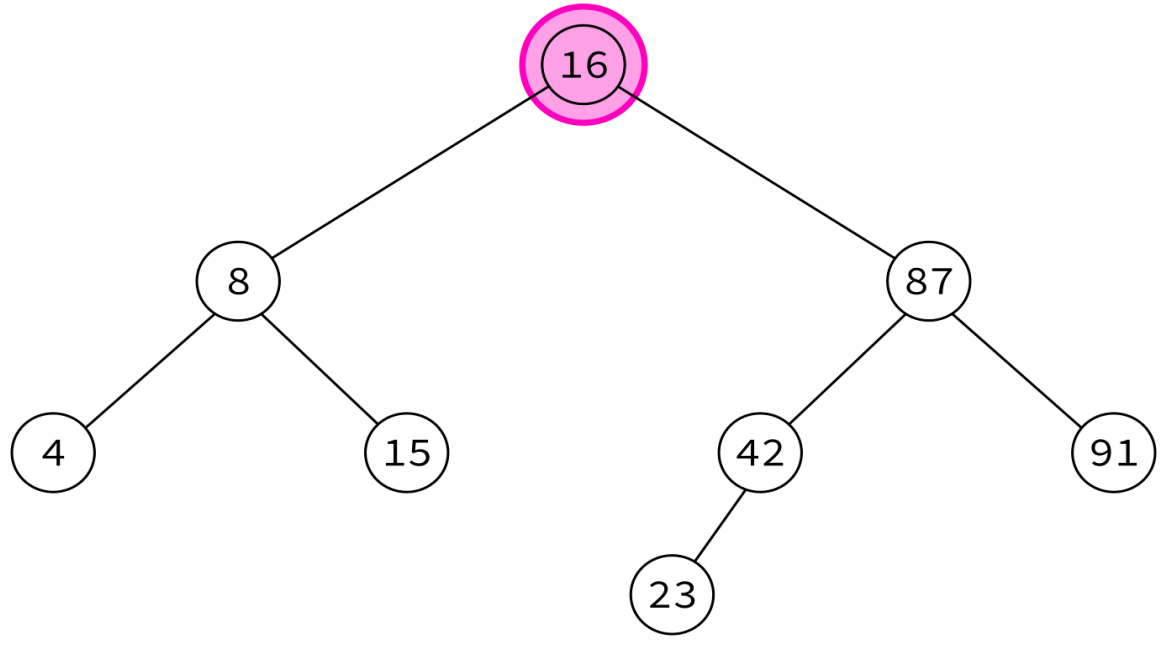
INORDER TRAVERSAL




```
4, 8, 15,
```

left, **node**, right

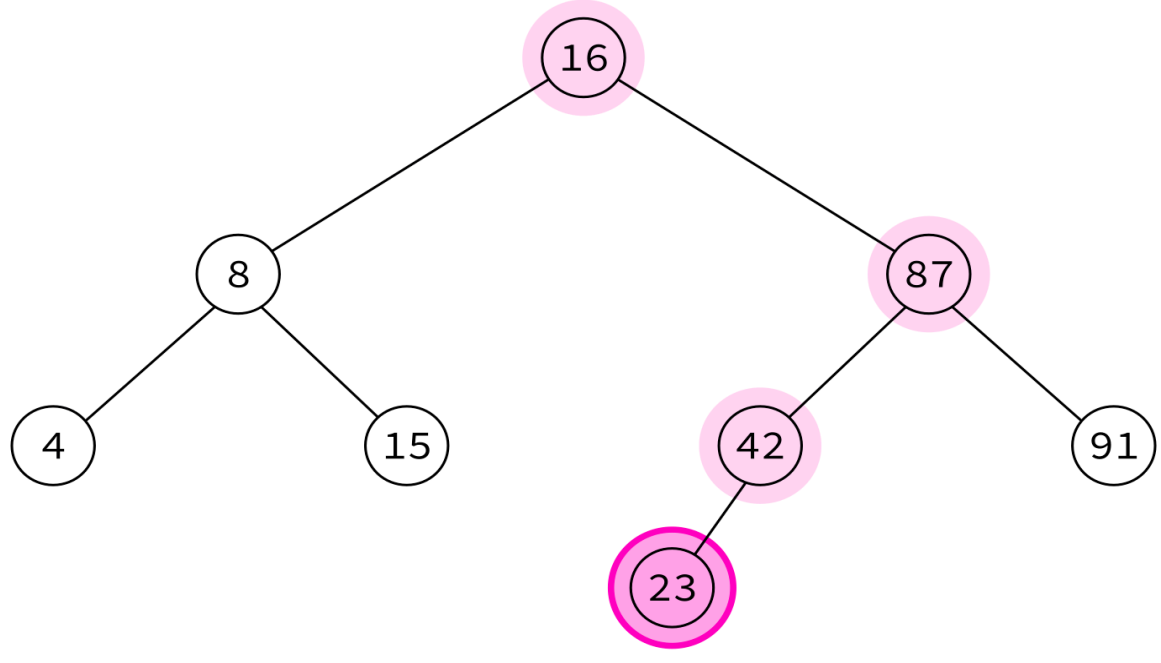
INORDER TRAVERSAL




```
4, 8, 15, 16,
```

left, **node**, right

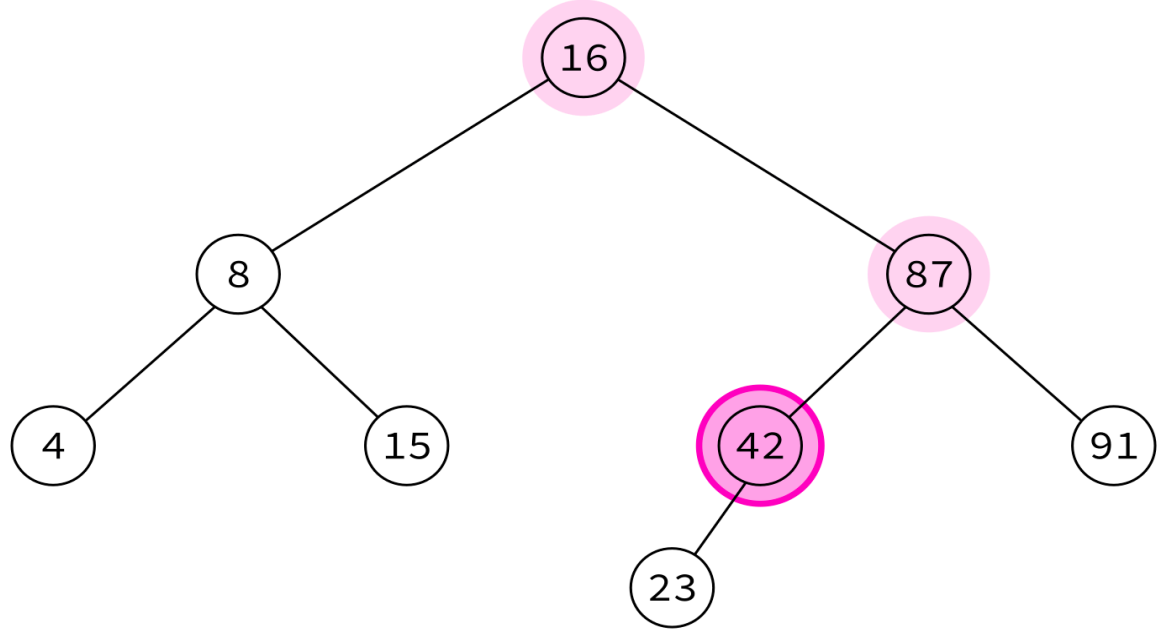
INORDER TRAVERSAL




```
4, 8, 15, 16, 23,
```

left, **node**, right

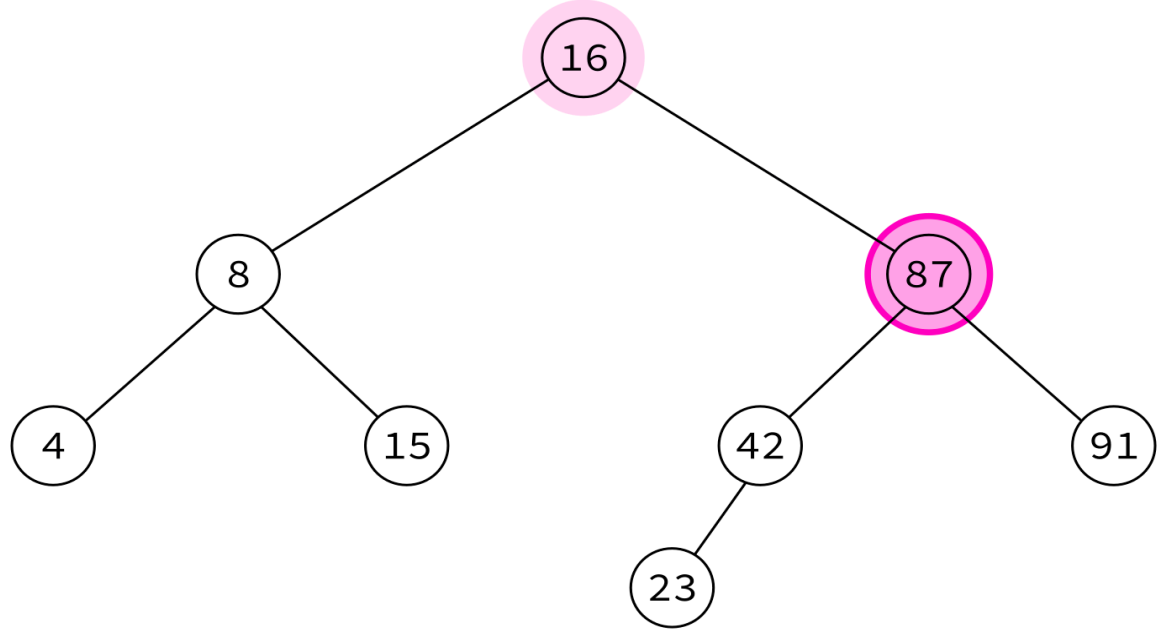
INORDER TRAVERSAL




```
4, 8, 15, 16, 23, 42,
```

left, **node**, right

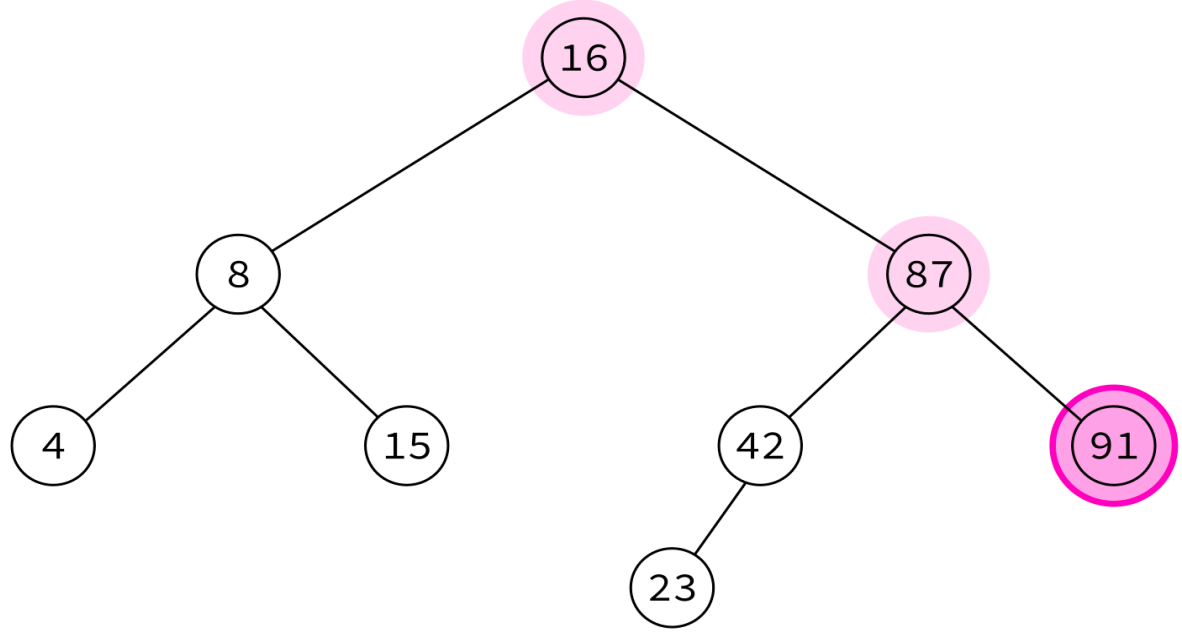
INORDER TRAVERSAL




```
4, 8, 15, 16, 23, 42, 87,
```

left, **node**, right

INORDER TRAVERSAL




```
4, 8, 15, 16, 23, 42, 87, 91
```

left, **node**, right

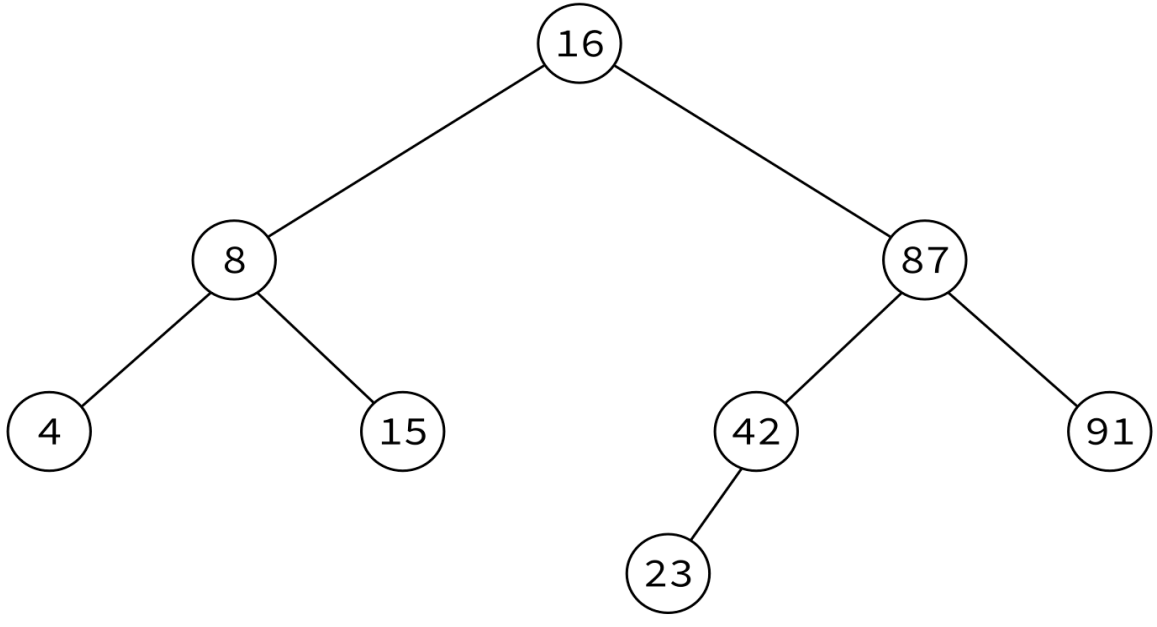
INORDER TRAVERSAL

Typical use: Turn a BST into a sorted list of keys.

LAST WORDS ON BINARY TREES

- BSTS make a lot of data accessible in a few "hops" from the root.
- They are a good choice for mutable data structures involving search operations.
- Deletion of a node is an important feature we didn't implement. (Take MCS 360!)

- Unbalanced trees are less efficient.

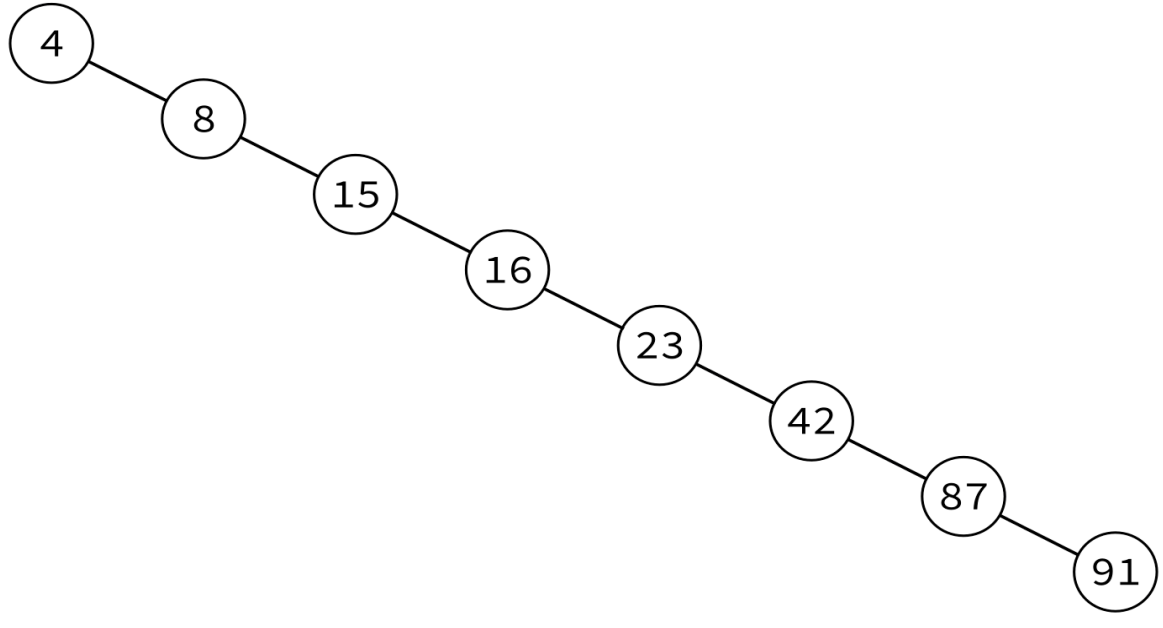


Balanced

depth ≈ $\log_2$(number of nodes)

MCS 360 usually covers rebalancing operations.

- Unbalanced trees are less efficient.



Unbalanced
depth ≈ number of nodes

MCS 360 usually covers rebalancing operations.

SET

Python's built-in type `set` represents an unordered collection of distinct objects.

Elements can be any hashable object. This is the same restriction as `dict` keys. E.g.

Allowed: `bool, int, float, str, tuple`

Not allowed: `list, set`

SET USAGE

```
S = { 4, 8, 15, 16, 23, 42 } # Set literal
S = set()  # New empty set
S.add(5)   # S is {5}
S.add(10)  # S is {5,10}
8 in S   # False
5 in S   # True
S.discard(1)  # Does nothing
S.remove(1)   # Raises KeyError
S.remove(5)   # Now S is {10}
S.pop()  # Remove and return one element
for x in S:  # sets are iterable
    print(x)
```

SET OPERATIONS

Binary operations returning new sets:

```
S | S2  # Evaluates to union of sets
S & S2  # Evaluates to intersection of sets
S.union(iterable)        # Like | but allows any iterable
S.intersection(iterable) # Like & but allows any iterable
```

Note: An earlier version of the slides claimed incorrectly that `&` and `|` allow arbitrary iterables. The methods `.union` and `.intersection` allow this, but the overloaded operators require the other operand to also be a set.

SET MUTATIONS

Operations that modify a set `S` based on contents of another collection.

```
# adds elements of iterable to S
S.update(iterable)

# remove anything from S that is NOT in the iterable
S.intersection_update(iterable)

# remove anything from S that is in the iterable
S.difference_update(iterable)
```

MORE ABOUT SET

`set` has lots of other features that are described in the documentation.

Python's `set` is basically a dictionary without values.

For large collections, it is much faster than using a list.

Appropriate whenever order is not important, and items cannot appear multiple times.

HISTOGRAM

You want to know how many times each character appears in a string.

```
hist = dict()
for c in s:
    hist[c] += 1
```

This won't work. Why?

DEFAULTDICT

Built-in module `collections` contains a class `defaultdict` that works like a dictionary, but if a key is requested that doesn't exist, it creates it and assigns a default value.

```
import collections
hist = collections.defaultdict(int)
for c in s:
    hist[c] += 1
```

This works!

The `defaultdict` constructor takes one argument, a function `default_factory`.

`default_factory` is called to make default values for keys when needed.

Common examples with built-in factories:

```
defaultdict(list)  # default value [] as returned by list()
defaultdict(int)   # default value 0, as returned by int()
defaultdict(float) # default value 0.0, as returned by float()
defaultdict(str)   # default value "", as returned by str()
```

REFERENCES

- In optional course texts:
  - Problem Solving with Algorithms and Data Structures using Python by Miller and Ranum, discusses binary trees in Chapter 7.
  - Lutz discusses sets in Chapter 5, in the subsection "Other Numeric Types" (even though there is nothing "numeric" about sets).
- Elsewhere:
  - Cormen, Leiserson, Rivest, and Stein discusses graph theory and trees in Appendices B.4 and B.5, and binary search trees in Chapter 12.

REVISION HISTORY

- 2021-03-11 Correction: set operations & and | don't allow non-set iterables
- 2021-03-02 Initial publication