

LECTURE 19

SET AND DEFAULTDICT

MCS 275 Spring 2023
Emily Dumas

LECTURE 19: SET AND DEFAULTDICT

Reminders and announcements:

- Project 1 graded. Check the feedback!
- Project 2 grading underway.
- Project 3 (due March 17) coming soon.
- Homework 7 due tomorrow, now accepting submissions.

PLAN

- Wrap up trees unit
- Start language features unit

INTEGERSET VARIANTS

- IntegerSet - uses BST
- IntegerSetUL - uses unsorted list
- IntegerSetSL - uses sorted list

INTEGERSET TIMING

`integerset.py` has been updated with a script to test addition and membership test times for 20,000 integers.

TRAVERSALS

Last time we introduced the **preorder**, **postorder**, and **inorder** traversals of a binary tree.

The `trees` module now has methods for each of these.

UNIQUELY DESCRIBING A TREE

Many different binary trees can have the same inorder traversal.

Many different binary trees can have the same preorder traversal.

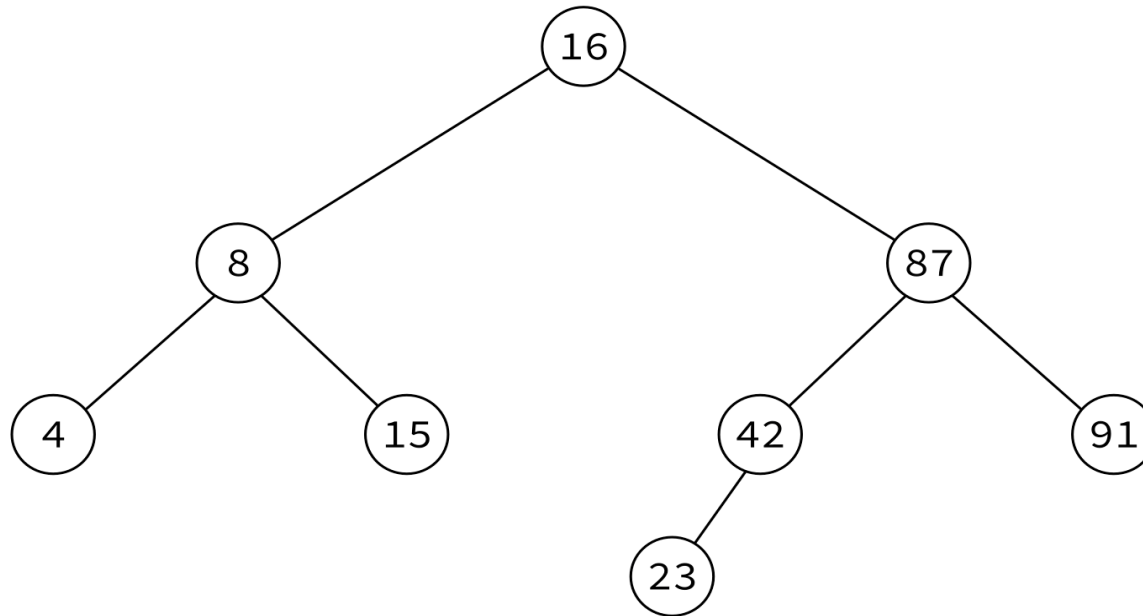
And yet:

Theorem: A binary tree T is uniquely determined by its inorder *and* preorder traversals.

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!)

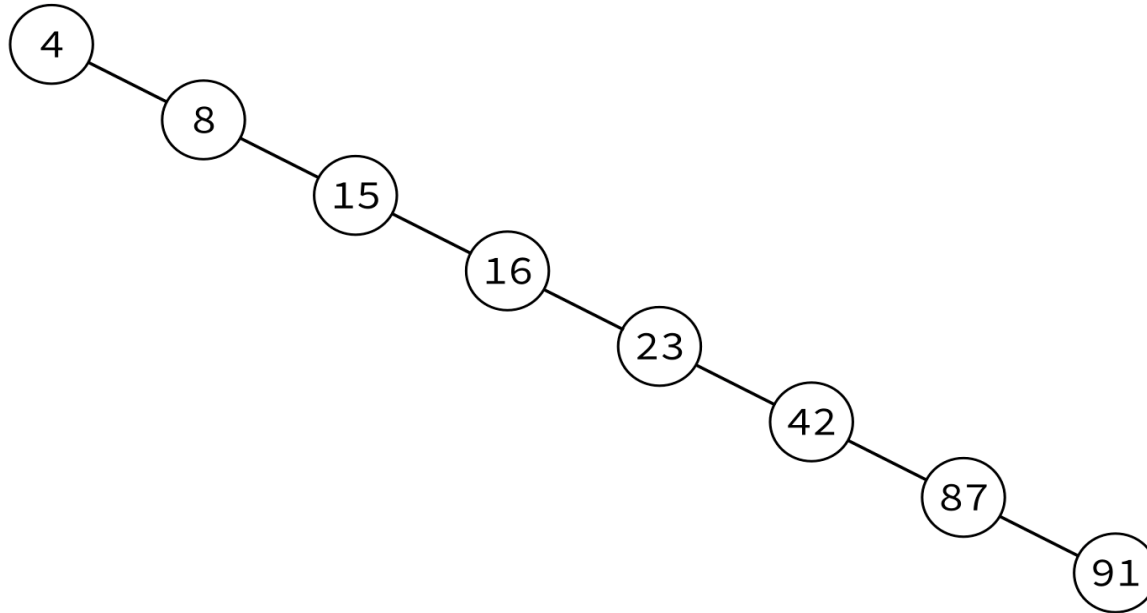
- Unbalanced trees are less efficient.



Balanced
 $\text{depth} \approx \log_2(\text{number of nodes})$

MCS 360 usually covers rebalancing operations.

- Unbalanced trees are less efficient.



Unbalanced
depth \approx number of nodes

MCS 360 usually covers rebalancing operations.

SET

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

You can put an object in a `set` if (and only if) it's allowed as a key of a `dict`. For built-in types that usually just means immutable.

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 (unclear which!)
for x in S: # sets are iterable (but no control over order)
    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
```

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

- 2022-03-02 Last year's lecture on this topic finalized
- 2022-02-26 Updated for 2023

