

CS 10: Problem solving via Object Oriented Programming

Info Retrieval

ADT Overview

List	
Description	Keep items stored in order by index
Common use	<ul style="list-style-type: none">• Grow to hold any number of items
Implementation options	<ul style="list-style-type: none">• Linked list• Growing array
Java provided	<ul style="list-style-type: none">• LinkedList• ArrayList

ADT Overview

	List	(Binary) Tree
Description	Keep items stored in order by index	Keep hierarchical relationship between nodes
Common use	<ul style="list-style-type: none">Grow to hold any number of items	<ul style="list-style-type: none">Find items quickly by Key<u>Generally</u> faster than List
Implementation options	<ul style="list-style-type: none">Linked listGrowing array	<ul style="list-style-type: none">BinaryTreeBST
Java provided	<ul style="list-style-type: none">LinkedListArrayList	

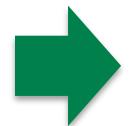
ADT Overview

	List	(Binary) Tree	Set
Description	Keep items stored in order by index	Keep hierarchical relationship between nodes	Keep an unordered set of objects
Common use	<ul style="list-style-type: none">Grow to hold any number of items	<ul style="list-style-type: none">Find items quickly by Key<u>Generally</u> faster than List	<ul style="list-style-type: none">Prevent duplicates
Implementation options	<ul style="list-style-type: none">Linked listGrowing array	<ul style="list-style-type: none">BinaryTreeBST	<ul style="list-style-type: none">ListBSTHash table
Java provided	<ul style="list-style-type: none">LinkedListArrayList		<ul style="list-style-type: none">TreeSetHashSet

ADT Overview

	List	(Binary) Tree	Set	Map
Description	Keep items stored in order by index	Keep hierarchical relationship between nodes	Keep an unordered set of objects	Keep a set of Key/Value pairs
Common use	<ul style="list-style-type: none">Grow to hold any number of items	<ul style="list-style-type: none">Find items quickly by Key<u>Generally</u> faster than List	<ul style="list-style-type: none">Prevent duplicates	<ul style="list-style-type: none">Find items quickly by Key
Implementation options	<ul style="list-style-type: none">Linked listGrowing array	<ul style="list-style-type: none">BinaryTreeBST	<ul style="list-style-type: none">ListBSTHash table	<ul style="list-style-type: none">ListBSTHash table
Java provided	<ul style="list-style-type: none">LinkedListArrayList		<ul style="list-style-type: none">TreeSetHashSet	<ul style="list-style-type: none">TreeMapHashMap

Agenda



1. Set ADT
2. Map ADT
3. Reading from file/keyboard
4. Search

Sets are an unordered collection of items without duplicates

Set ADT

- Model for mathematical definition of a Set
- Like a List, but:
 - Unordered (no i^{th} item, can't set/get by position)
 - No duplicates allowed
- Operations:
 - $\text{add}(E e)$ – adds e to Set if not already present
 - $\text{contains}(E e)$ – returns true if e in Set, else false
 - $\text{isEmpty}()$ – true if no elements in Set, else false
 - $\text{Iterator}<E> \text{ iterator}()$ – returns iterator over Set
 - $\text{remove}(E e)$ – removes e from Set
 - $\text{size}()$ – returns number of elements in Set

Sets start out empty

Initial state

Set

isEmpty: True
size: 0

First item added will always create a new entry in the Set (item can't be a duplicate)

add(1)

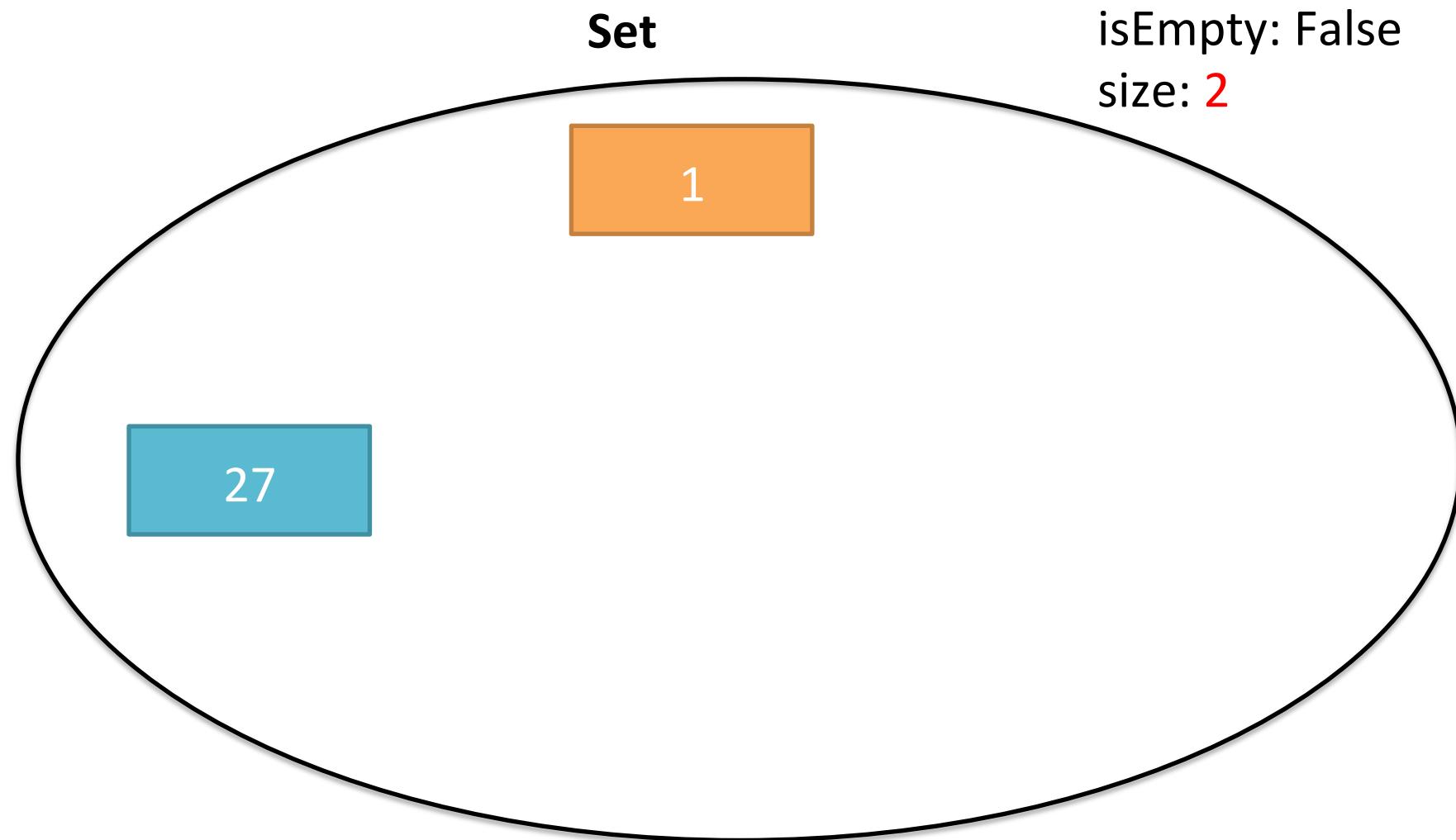
Set



isEmpty: **False**
size: **1**

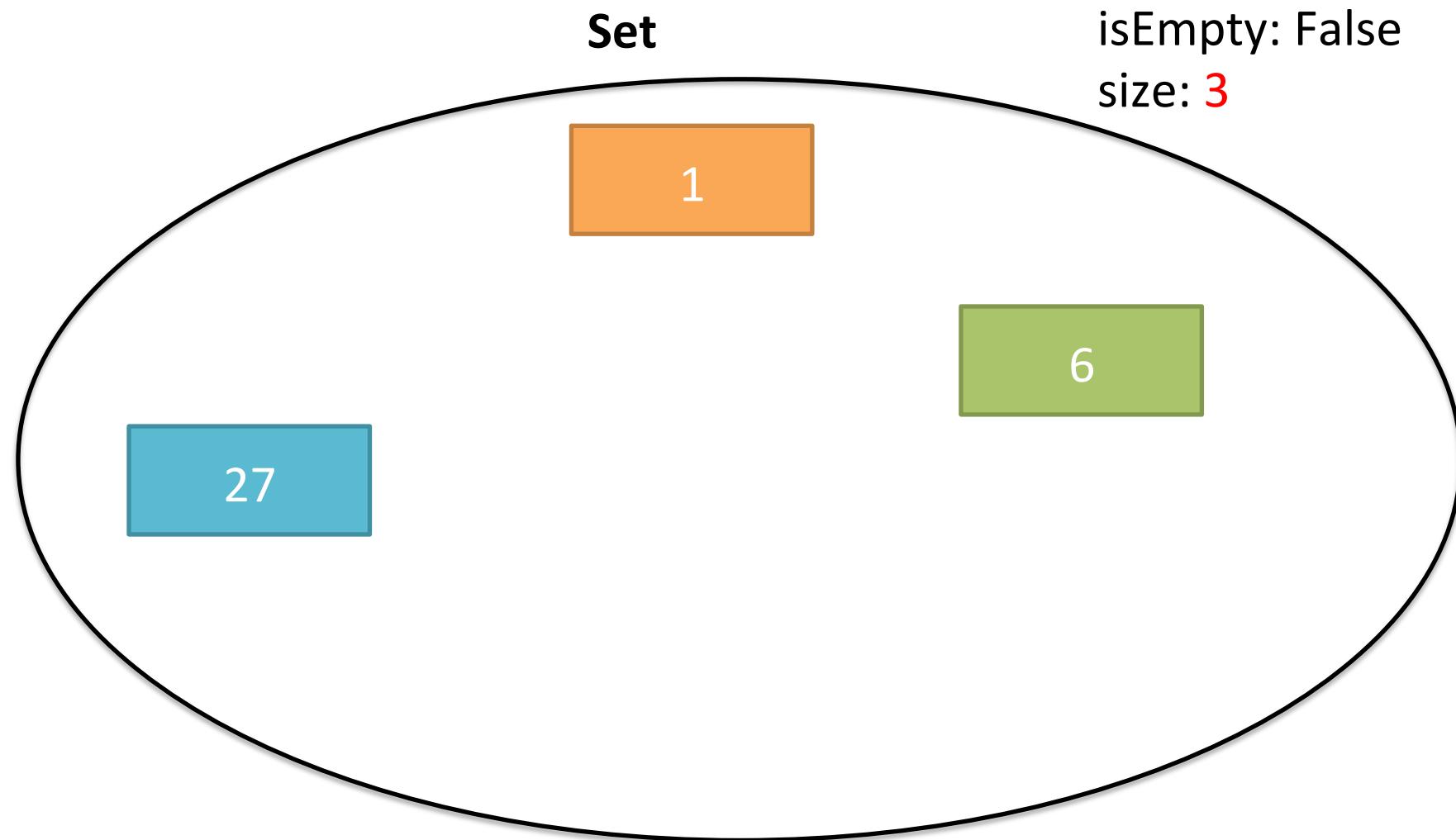
Can think of adding items to Set like adding items to “Bag of items” – no item ordering

add(27)



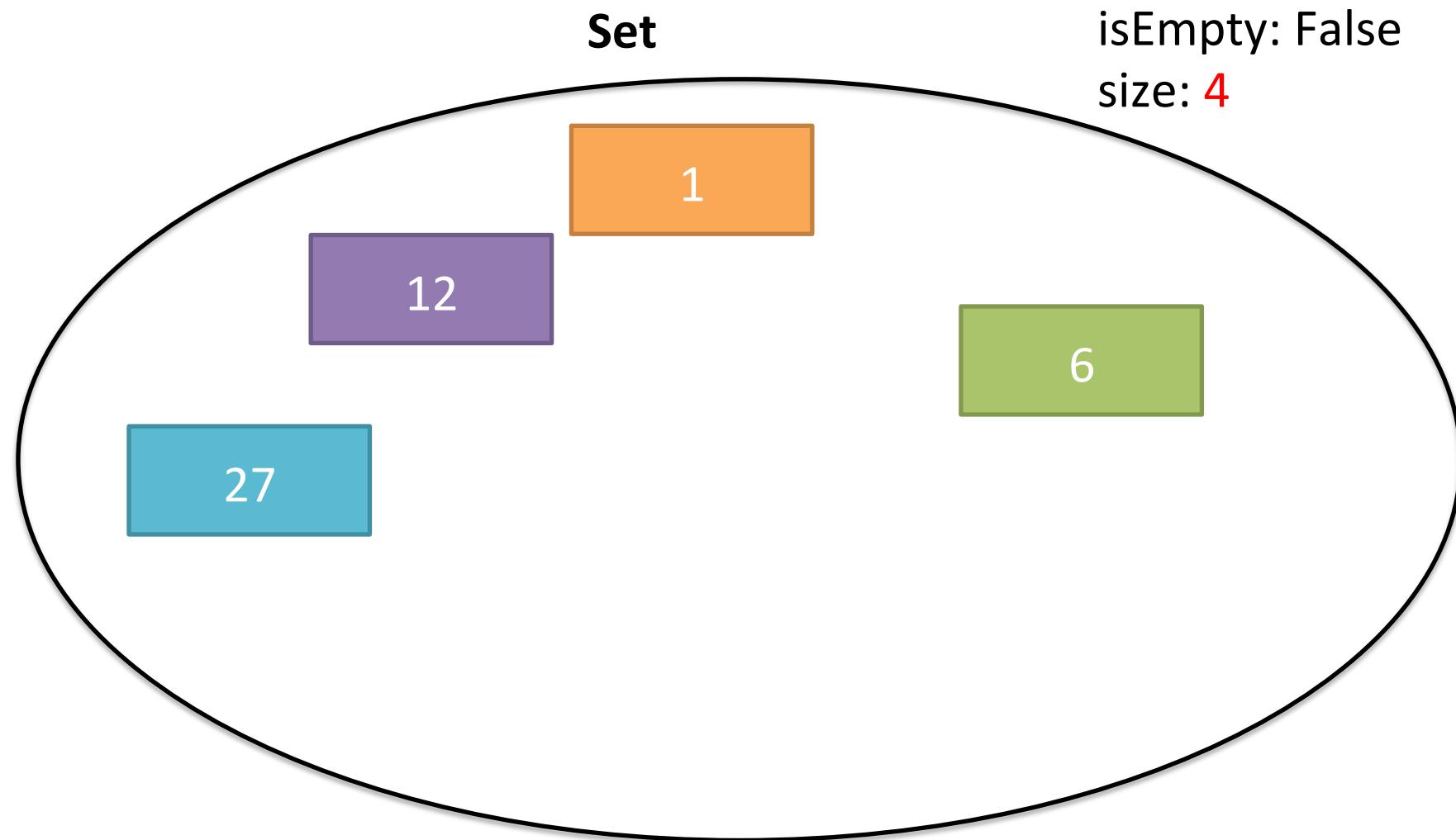
Can think of adding items to Set like adding items to “Bag of items” – no item ordering

add(6)



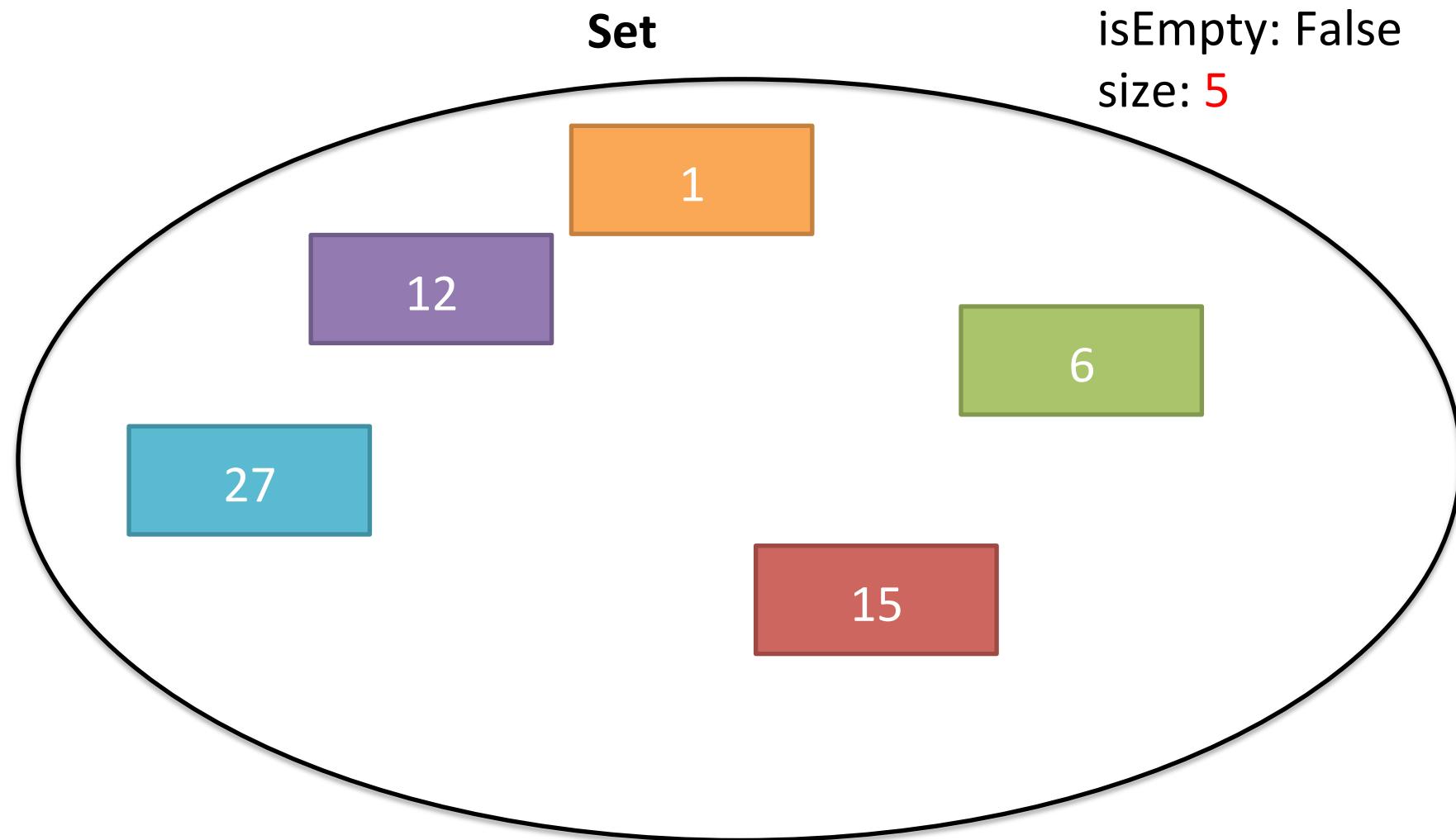
Can think of adding items to Set like adding items to “Bag of items” – no item ordering

add(12)



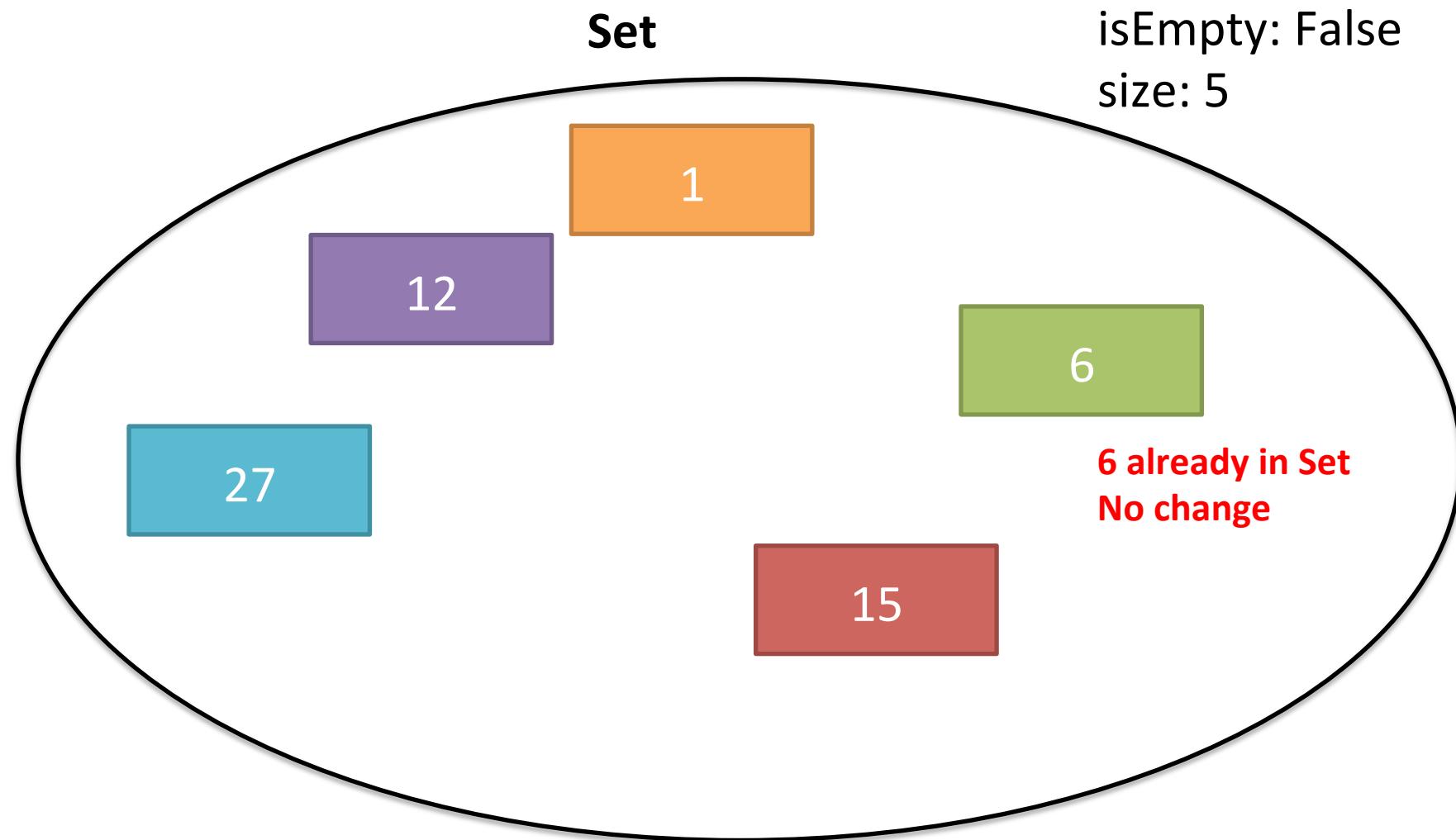
Can think of adding items to Set like adding items to “Bag of items” – no item ordering

add(15)



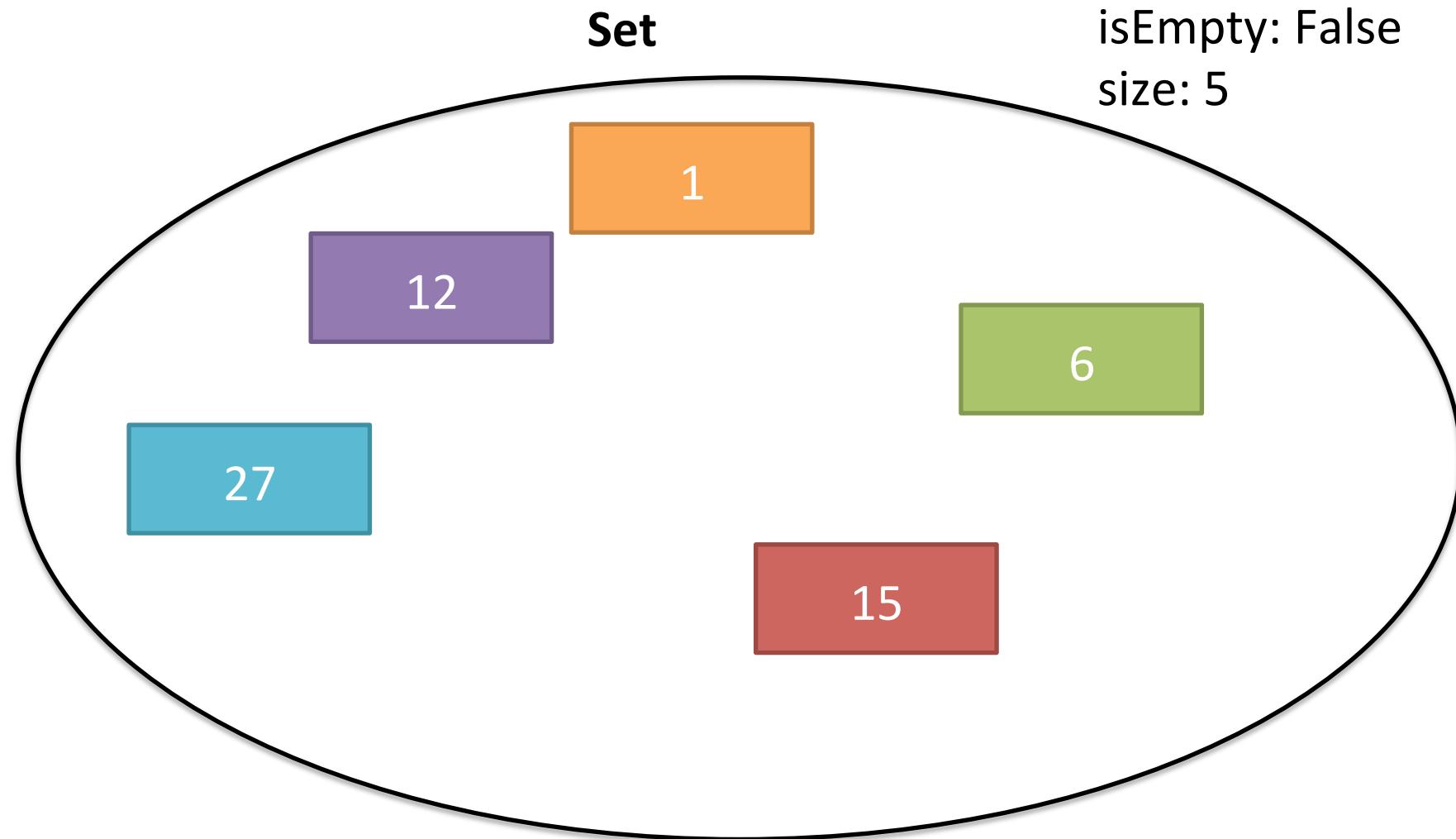
Adding an item that is already in the Set does not change the Set

add(6)



Items can be removed

remove(1)



Items can be removed

remove(1)

Set

isEmpty: False
size: 4

1 removed
size reduced

12

6

27

15

Can also check to see if item is in Set

contains(12)

True **Set** **isEmpty: False**
size: 4



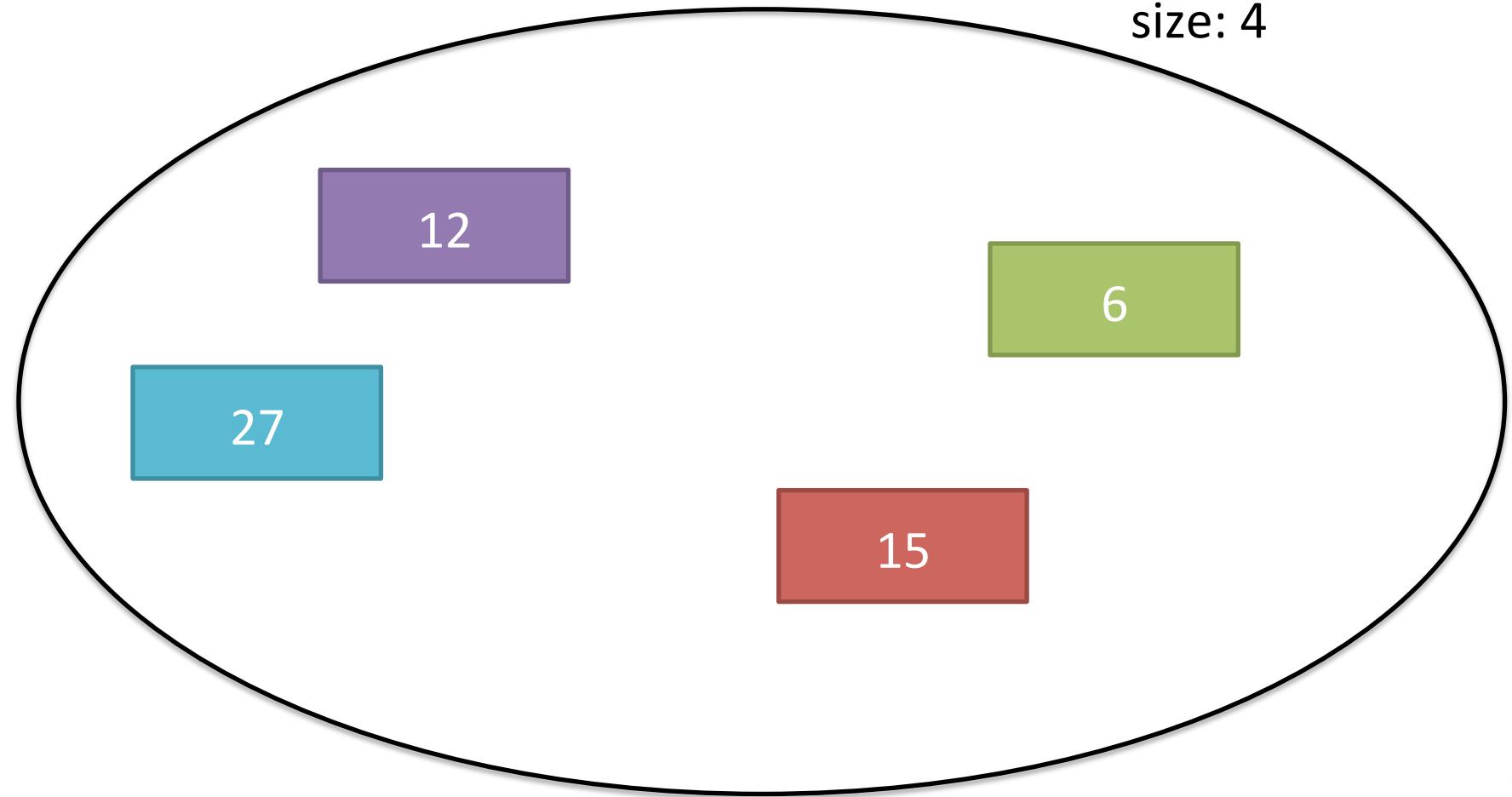
Can also check to see if item is in Set

contains(13)

False

Set

isEmpty: False
size: 4



12

27

15

6

Trees are one way to implement the Set ADT

Sets implemented with Trees

- Could implement as a List, but linear search time
- Trees are a natural way to think about implementation
- If the Set is implemented with a Binary Search Tree (BST)

Trees are one way to implement the Set ADT

Sets implemented with Trees

- Could implement as a List, but linear search time
- Trees are a natural way to think about implementation
- If the Set is implemented with a Binary Search Tree (BST)

Operation	Run-time	Notes
$\text{add}(e)$	$O(h)$	<ul style="list-style-type: none">• Search for node until found or hit leaf• If not found, add new leaf (if found do nothing)• Might have to add node on longest path• Can't be more than $h+1$ checks

Trees are one way to implement the Set ADT

Sets implemented with Trees

- Could implement as a List, but linear search time
- Trees are a natural way to think about implementation
- If the Set is implemented with a Binary Search Tree (BST)

Operation	Run-time	Notes
$\text{add}(e)$	$O(h)$	<ul style="list-style-type: none">• Search for node until found or hit leaf• If not found, add new leaf (if found do nothing)• Might have to add node on longest path• Can't be more than $h+1$ checks
$\text{contains}(e)$	$O(h)$	<ul style="list-style-type: none">• Search for node until found or hit leaf• Might have to search longest path• Can't be more than $h+1$ checks

Trees are one way to implement the Set ADT

Sets implemented with Trees

- Could implement as a List, but linear search time
- Trees are a natural way to think about implementation
- If the Set is implemented with a Binary Search Tree (BST)

Operation	Run-time	Notes
$\text{add}(e)$	$O(h)$	<ul style="list-style-type: none">• Search for node until found or hit leaf• If not found, add new leaf (if found do nothing)• Might have to add node on longest path• Can't be more than $h+1$ checks
$\text{contains}(e)$	$O(h)$	<ul style="list-style-type: none">• Search for node until found or hit leaf• Might have to search longest path• Can't be more than $h+1$ checks
$\text{remove}(e)$	$O(h)$	<ul style="list-style-type: none">• Traverse tree to find element, then delete it

Trees are one way to implement the Set ADT

Sets implemented with Trees

- Could implement as a List, but linear search time
- Trees are a natural way to think about implementation
- If the Set is implemented with a Binary Search Tree (BST)

Operation	Run-time	Notes
$add(e)$	$O(h)$	<ul style="list-style-type: none">• Search for node until found or hit leaf• If not found, add new leaf (if found do nothing)• Might have to add node on longest path• Can't be more than $h+1$ checks
$contains(e)$	$O(h)$	<ul style="list-style-type: none">• Search for node until found or hit leaf• Might have to search longest path• Can't be more than $h+1$ checks
$remove(e)$	$O(h)$	<ul style="list-style-type: none">• Traverse tree to find element, then delete it

- Soon we will see another, more efficient way to implement a Set using a hash table

Can use a Set to easily identify the unique words in a body of text

Text from which to identify unique words

"Pretend that this string was loaded from a web page. We won't go to all that trouble here. This string contains multiple words. And multiple copies of multiple words. And multiple words with multiple copies. It is to be used as a test to demonstrate how sets work in removing redundancy by keeping only one copy of each thing. Is it very very redundant in having more than one copy of some words?"

Pseudocode

- Create Set with String as element
- Loop over each word in text
- Add to Set
- Print Set when done

Set <String>

- Add each word in text to Set
- Duplicates not maintained

Can use a Set to easily identify the unique words in a body of text

Text from which to identify unique words

"**Pretend** that this string was loaded from a web page. We won't go to all that trouble here. This string contains multiple words. And multiple copies of multiple words. And multiple words with multiple copies. It is to be used as a test to demonstrate how sets work in removing redundancy by keeping only one copy of each thing. Is it very very redundant in having more than one copy of some words?"

Pseudocode

- Create Set with String as element
- Loop over each word in text
- Add to Set
- Print Set when done

Set <String>

Pretend

Can use a Set to easily identify the unique words in a body of text

Text from which to identify unique words

"Pretend **that** this string was loaded from a web page. We won't go to all that trouble here. This string contains multiple words. And multiple copies of multiple words. And multiple words with multiple copies. It is to be used as a test to demonstrate how sets work in removing redundancy by keeping only one copy of each thing. Is it very very redundant in having more than one copy of some words?"

Pseudocode

- Create Set with String as element
- Loop over each word in text
- Add to Set
- Print Set when done

Set <String>

Pretend
that

Can use a Set to easily identify the unique words in a body of text

Text from which to identify unique words

"Pretend that this string was loaded from a web page. We won't go to all **that** trouble here. This string contains multiple words. And multiple copies of multiple words. And multiple words with multiple copies. It is to be used as a test to demonstrate how sets work in removing redundancy by keeping only one copy of each thing. Is it very very redundant in having more than one copy of some words?"

Pseudocode

- Create Set with String as element
- Loop over each word in text
- Add to Set
- Print Set when done

Set <String>

Pretend

that

this

string

was

loaded

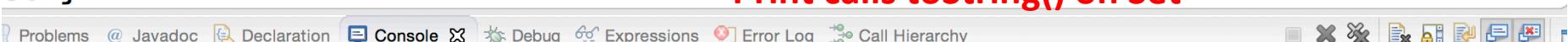
...

- "that" seen again
- Already in Set, so Set does not change
- At the end the Set will contain all the unique words in the text

UniqueWords.java: Use a Set to easily identify the unique words in a body of text

```
10 public static void main(String[] args) { Large amount of text simulates webpage
11     String page = "Pretend that this string was loaded from a web page. We
12     split() makes an array + won't go to all that trouble here. This string contains mult
13     with entry for each + "words. And multiple copies of multiple words. And multiple "
14     word (including + "words with multiple copies. It is to be used as a test to "
15     duplicates) + "demonstrate how sets work in removing redundancy by keeping c
16     String[] allWords = page.split("[ .,?!]+"); // split on punctuation and
17
18     // Declare new Set to hold unique words
19     Set<String> uniqueWords = new TreeSet<String>(); Java has Set implementation
20     based on Tree
21     Add all words to Set, discarding duplicates Implements Set interface
22     // Loop over all the words split out of the string, adding to set
23     for (String s: allWords) {
24         uniqueWords.add(s.toLowerCase()); Set elements are Strings here
25     }
26
27     System.out.println(allWords.length + " words"); //note: this is not the
28     System.out.println(uniqueWords.size() + " unique words"); //this is the
29     System.out.println(uniqueWords); //print the unique words
30 }
```

No duplicate words
Print calls `toString()` on Set



terminated> UniqueWords [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_112.jdk/Contents/Home/bin/java (Jan 2, 2018, 6:59:33 AM)

a, all, and, as, be, by, contains, copies, copy, demonstrate, each, from, go, having

Agenda

1. Set ADT
2. Map ADT
3. Reading from file/keyboard
4. Search

Map ADT associates Keys with Values

Map ADT

- Key is used to look up a Value (ex., student ID finds student record)
- Python programmers can think of Maps as Dictionaries
- Value could be an object (e.g., a person object or student record containing courses taken and grades for each)
- Operations:
 - *containsKey (K key)* – true if *key* in Map, else false
 - *containsValue (V value)* – true if one or more Keys contain *value*
 - *get (K key)* – returns Value for specified *key* or null
 - *isEmpty ()* – true if no elements in Map, else false
 - *keySet ()* – returns Set of Keys in Map
 - *put (K key, V value)* – store *key/value* in Map; overwrite existing (NOTE: no add operation in Map ADT)
 - *remove (K key)* – removes *key* from Map and returns *value*
 - *size ()* – returns number of elements in Map

Like Sets, Maps initially start out empty

Map		isEmpty: True size: 0
Key <StudentID>	Value <Student Name>	

Items are adding to a Map using *put(Key, Value)*

put(123, "Charlie")

Map	
Key <StudentID>	Value <Student Name>
123	Charlie

isEmpty: **False**
size: **1**

Items are adding to a Map using *put(Key, Value)*

put(987, "Alice")

Map	
Key <StudentID>	Value <Student Name>
123	Charlie
987	Alice

isEmpty: False
size: 2

Items are adding to a Map using *put(Key, Value)*

put(456, "Bob")

Map	
Key <StudentID>	Value <Student Name>
123	Charlie
987	Alice
456	Bob

isEmpty: False
size: 3

Items are adding to a Map using *put(Key, Value)*

put(456, “Bob”)

Map	
Key <StudentID>	Value <Student Name>
123	Charlie
987	Alice
456	Bob

isEmpty: False
size: 3

Items are adding to a Map using *put(Key, Value)*

put(456, “Bob”)

Map	
Key <StudentID>	Value <Student Name>
123	Charlie
987	Alice
456	Bob

isEmpty: False
size: 3

- **NOTE: Keys are not necessarily kept in order**
- **Implementation details left to the designer**

If an item already exists, *put(Key, Value)* will update the Value for that Key

put(987, “Ally”)

Map		isEmpty: False size: 3
Key <StudentID>	Value <Student Name>	
123	Charlie	
987	Alice	
456	Bob	

If an item already exists, *put(Key, Value)* will update the Value for that Key

put(987, “Ally”)

Map		isEmpty: False size: 3
Key <StudentID>	Value <Student Name>	
123	Charlie	
987	Ally	
456	Bob	

Can remove items by Key and get Value for that Key (or null if Key not found)

remove(987) => “Ally”

Map		isEmpty: False size: 3
Key <StudentID>	Value <Student Name>	
123	Charlie	
987	Ally	
456	Bob	

Removes item with Key and returns Value

Can remove items by Key and get Value for that Key (or null if Key not found)

remove(987) => null

Map		isEmpty: False	size: 2
Key <StudentID>	Value <Student Name>		
123	Charlie		
456	Bob		

Returns null if Key not found

`keyset()` returns a Set of Keys in the Map

`keyset() => Set {123, 456}`

Map		isEmpty: False size: 2
Key <StudentID>	Value <Student Name>	
123	Charlie	
456	Bob	

get(Key) returns the Value for the Key (or null if Key not found)

get(456) => “Bob”

Map		isEmpty: False	size: 2
Key <StudentID>	Value <Student Name>		
123	Charlie		
456	Bob		

get(Key) returns the Value for the Key (or null if Key not found)

get(987) => null

Map		isEmpty: False	size: 2
Key <StudentID>	Value <Student Name>		
123	Charlie		
456	Bob		

containsKey(Key) returns True if Key in Map, False otherwise

containsKey(123) => True

Map		isEmpty: False	size: 2
Key <StudentID>	Value <Student Name>		
123	Charlie		
456	Bob		

containsKey(Key) returns True if Key in Map, False otherwise

containsKey(987) => False

Map		isEmpty: False	size: 2
Key <StudentID>	Value <Student Name>		
123	Charlie		
456	Bob		

containsValue(Value) returns True if Value in Map, False otherwise

containsValue("Bob") => True

Map		isEmpty: False size: 2
Key <StudentID>	Value <Student Name>	
123	Charlie	
456	Bob	

containsValue(Value) returns True if Value in Map, False otherwise

containsValue("Alice") => False

Map		isEmpty: False	size: 2
Key <StudentID>	Value <Student Name>		
123	Charlie		
456	Bob		

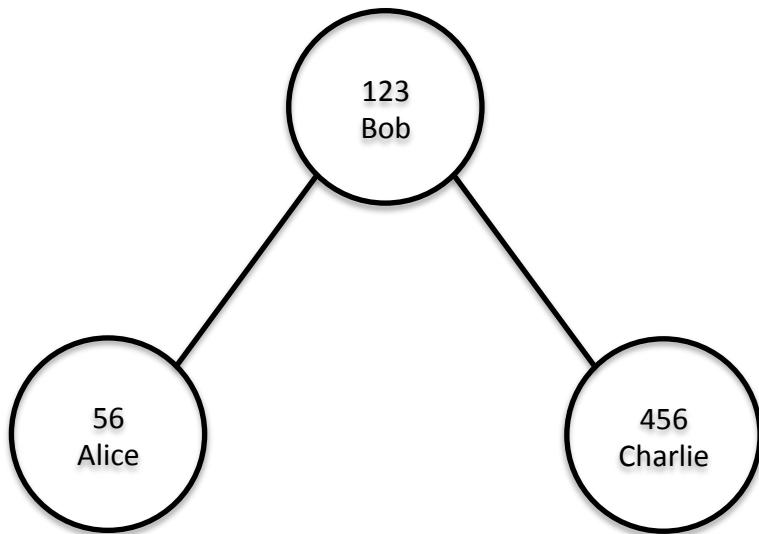
Trees are one way to implement the Map ADT

Maps implemented with Trees

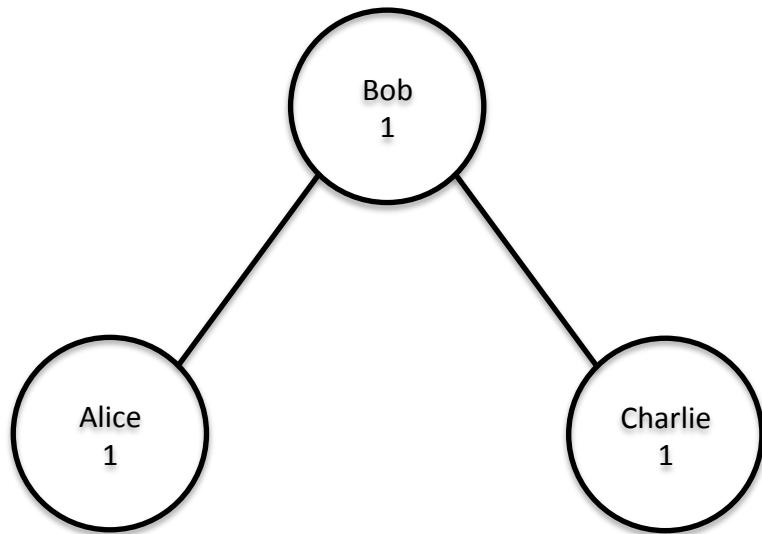
- Could implement as a List, but linear search time
- Like Sets, Trees are natural way to think about Map implementation
- Problem: no easy way to implement `containsValue()` because Tree searches for Keys not Values (but `containsKey()` is easy!)
 - Could search entire Tree for Value
 - Problem: linear time
 - Idea: keep a Set of values, update on each `put` and then search Set
 - Problem: the same Value could be stored with different keys, so if delete Key, can't necessarily delete Value from Set
 - Better idea: keep a second Tree with Values as Keys and counts of each Value
 - When adding a Value, increment its count in the second Tree
 - When deleting a Key, decrement Value count, delete Value in second Tree if count goes to zero
 - Now have $O(h)$ time search for `containsValue()`
 - Uses more memory, but has better speed

For *containsValue()* keep one Tree for Key/Value and another Tree with Values/count

Tree with Key and Value



Tree with Value and count



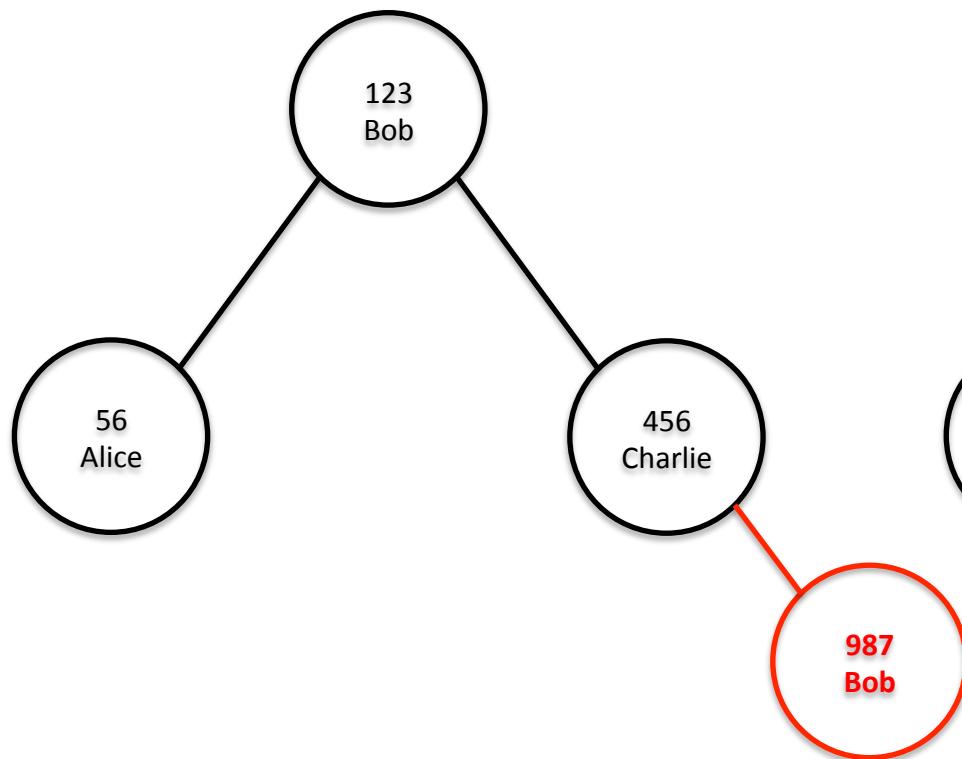
- Each node has Key and Value
- Duplicate Values allowed, duplicate Keys not allowed
- Easy to do *containsKey(key)*
 - Search Tree for *key*
 - Return false if hit leaf and *key* not found, else true

- Each node has Value and count of how many times Value in Map
- Easy to do *containsValue(value)*
 - Search Tree for *value*
 - Return false if hit leaf and *value* not found, else true
- **This trades memory for speed**

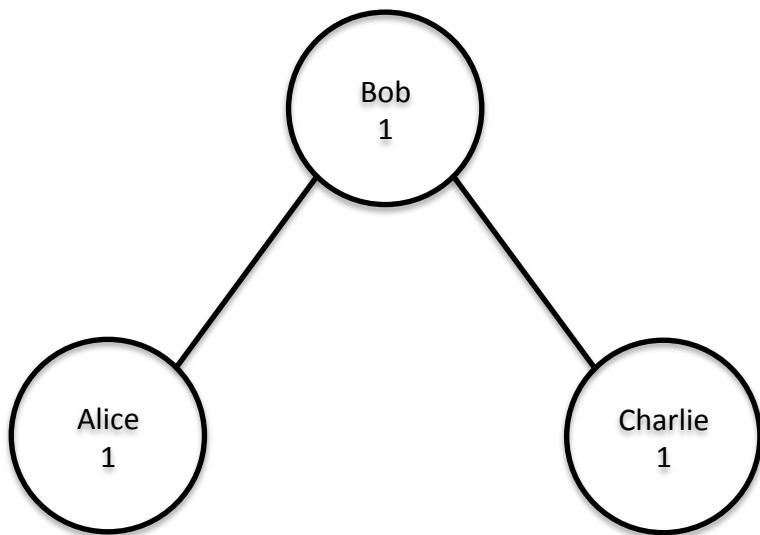
On $\text{put}(key, value)$, add Key/Value to Tree, increment count (if needed)

Put(987, "Bob")

Tree with Key and Value



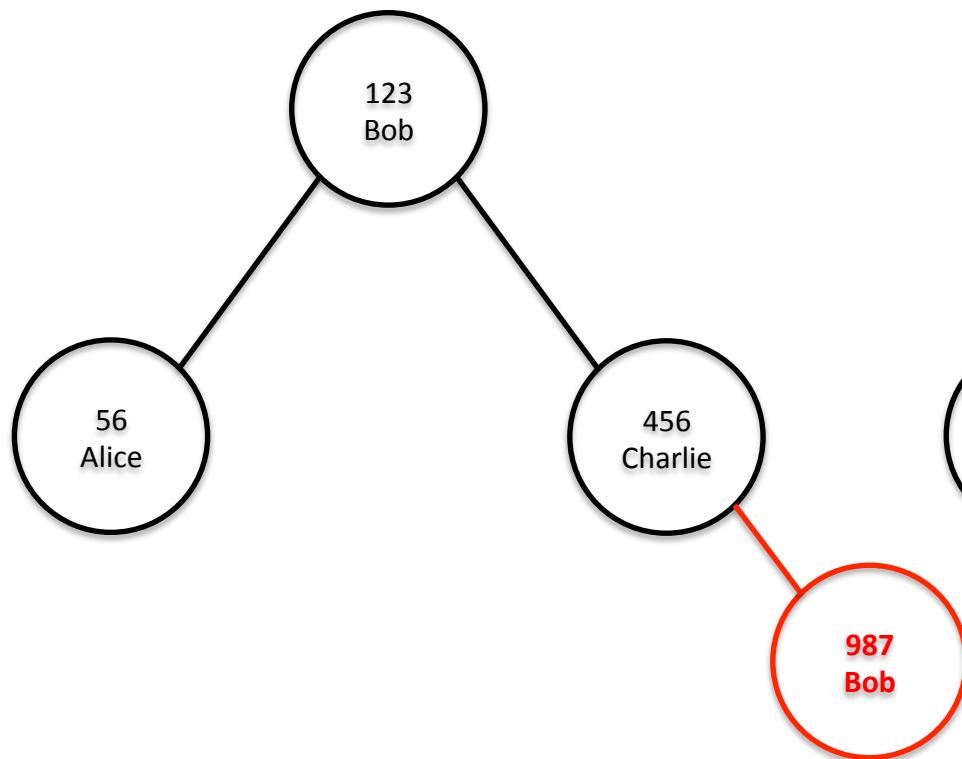
Tree with Value and count



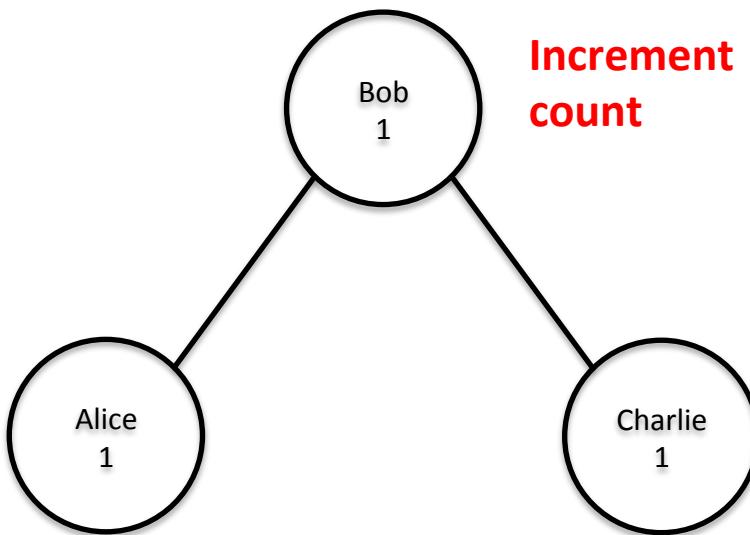
On $\text{put}(key, value)$, add Key/Value to Tree, increment count (if needed)

Put(987, "Bob")

Tree with Key and Value



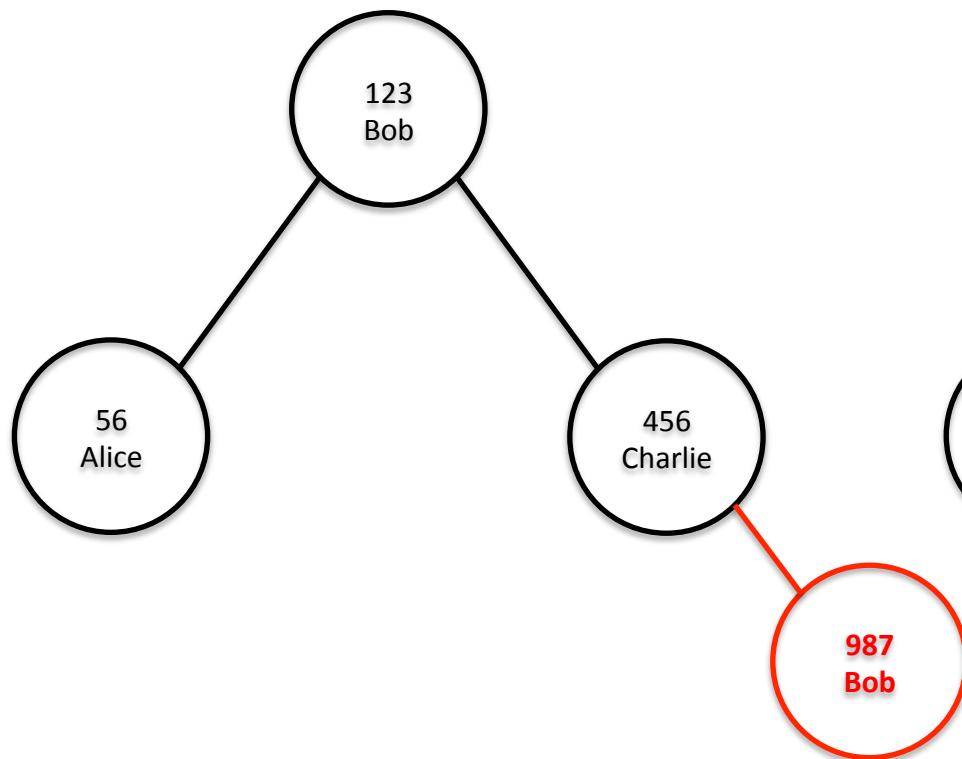
Tree with Value and count



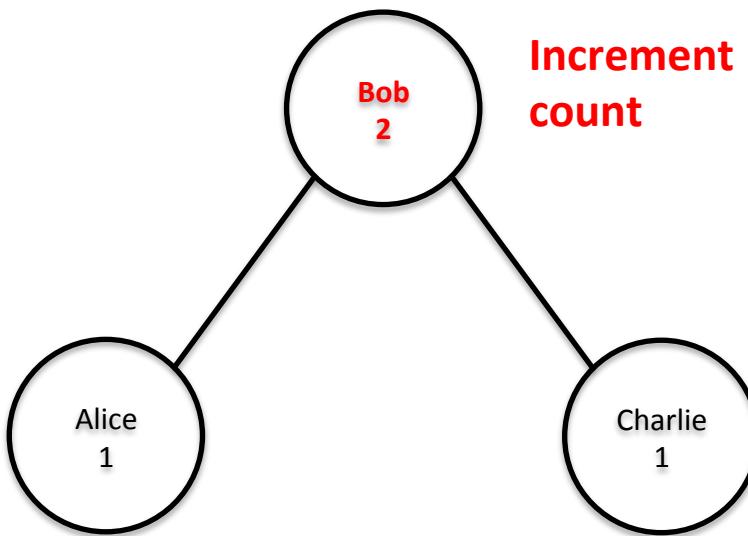
On $\text{put}(key, value)$, add Key/Value to Tree, increment count (if needed)

Put(987, "Bob")

Tree with Key and Value



Tree with Value and count

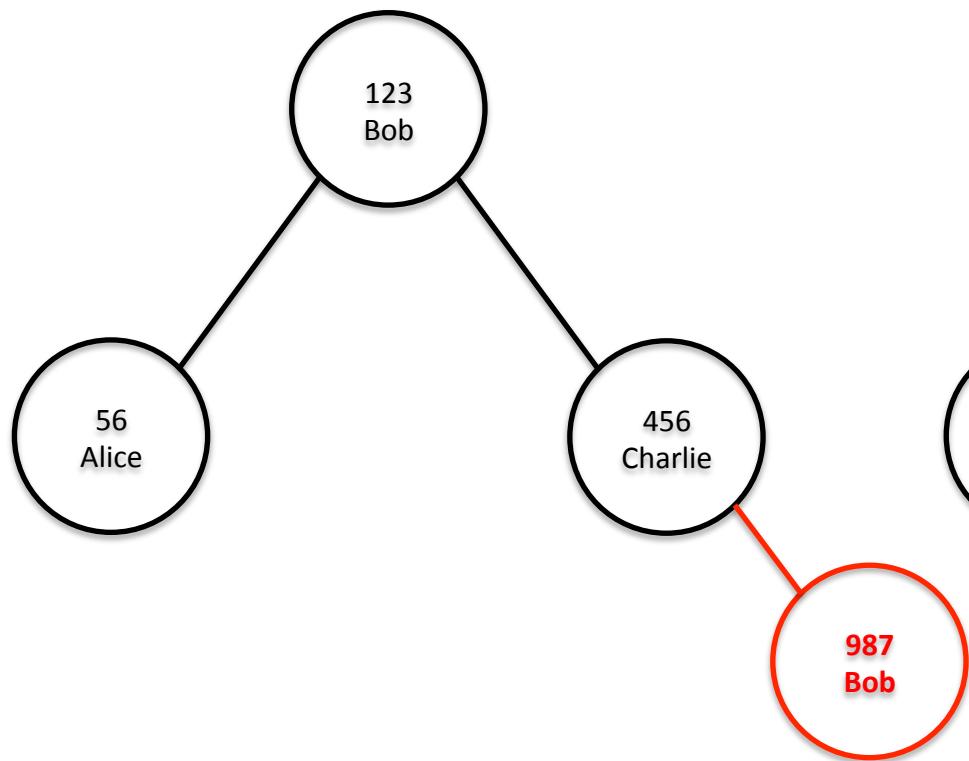


Increment
count

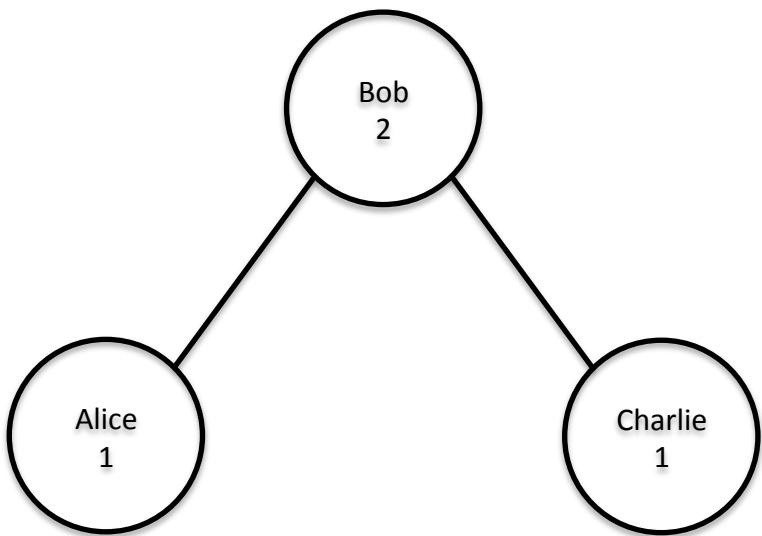
On *remove(key)*, delete Key/Value and decrement count

remove(987)

Tree with Key and Value



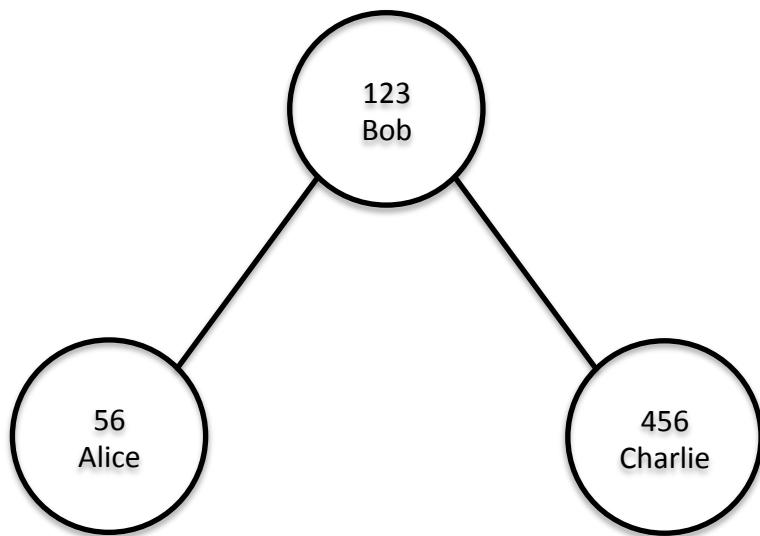
Tree with Value and count



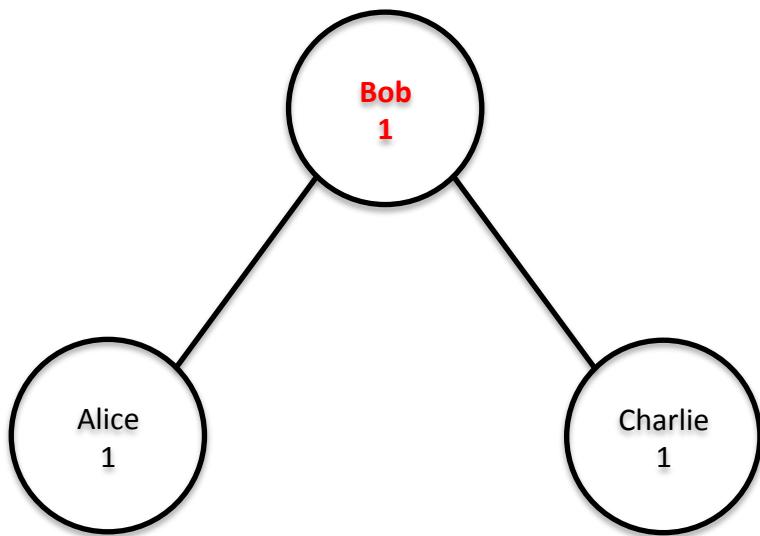
On *remove(key)*, delete Key/Value and decrement count

remove(987)

Tree with Key and Value



Tree with Value and count

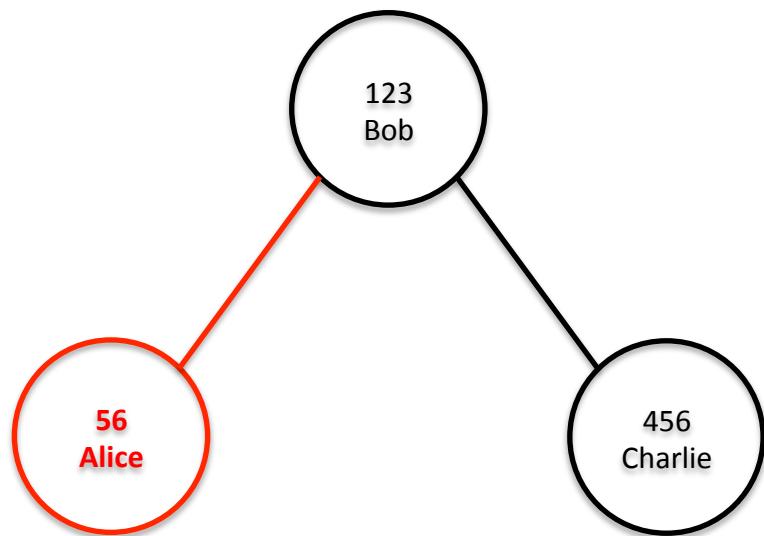


- Know there is still one “Bob” in the Tree
- Don’t delete node “Bob” from this tree

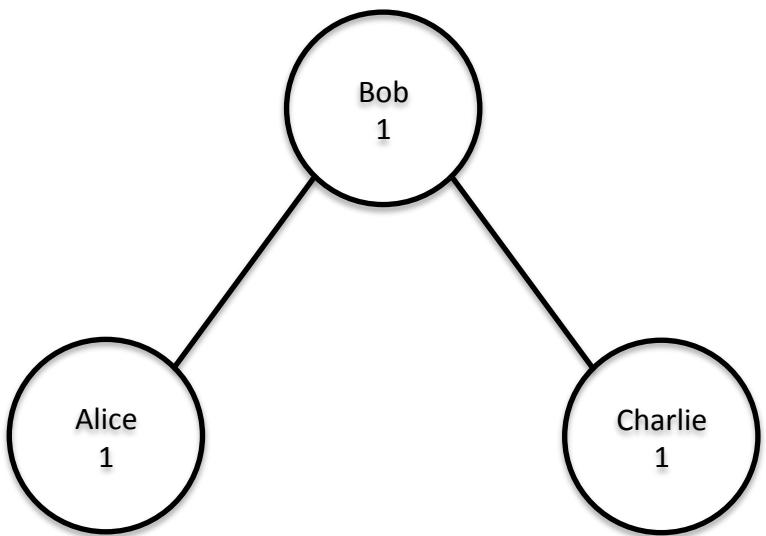
On *remove(key)*, delete Key/Value and decrement count

remove(56)

Tree with Key and Value



Tree with Value and count

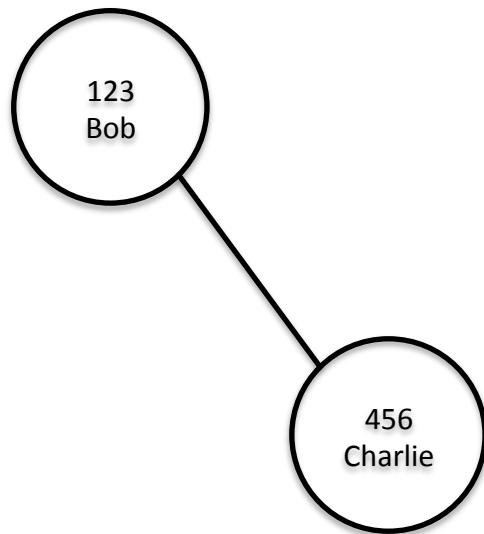


Remove “Alice”

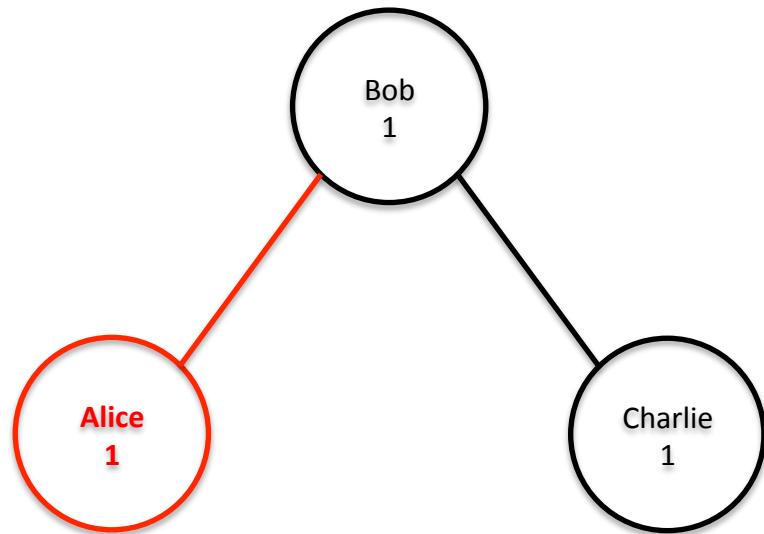
On *remove(key)*, delete Key/Value and decrement count

remove(56)

Tree with Key and Value



Tree with Value and count



**Because count goes to 0,
remove “Alice” here too**

**Must also update counts if
a *put()* replaces a value**

Can use a Map to count word occurrences in a body of text

Text from which to identify unique words

"**Pretend** that this string was loaded from a web page. We won't go to all that trouble here. This string contains multiple words. And multiple copies of multiple words. And multiple words with multiple copies. It is to be used as a test to demonstrate how sets work in removing redundancy by keeping only one copy of each thing. Is it very very redundant in having more than one copy of some words?"

Pseudocode

- Create Map with String Key and Integer Value
- Loop over each word in text
- If Map contains (word)
 - Increment count Value
 - Else put (word) with Value 1
- Print Map when done

Map	
Key <String>	Value <Integer>
Pretend	1

Can use a Map to count word occurrences in a body of text

Text from which to identify unique words

"Pretend **that** this string was loaded from a web page. We won't go to all that trouble here. This string contains multiple words. And multiple copies of multiple words. And multiple words with multiple copies. It is to be used as a test to demonstrate how sets work in removing redundancy by keeping only one copy of each thing. Is it very very redundant in having more than one copy of some words?"

Pseudocode

- Create Map with String Key and Integer Value
- Loop over each `word` in text
- If Map `contains(word)`
 - Increment count Value
 - Else `put(word)` with Value 1
- Print Map when done

Map	
Key <String>	Value <Integer>
Pretend	1
that	1

Can use a Map to count word occurrences in a body of text

Text from which to identify unique words

"Pretend that **this** string was loaded from a web page. We won't go to all that trouble here. This string contains multiple words. And multiple copies of multiple words. And multiple words with multiple copies. It is to be used as a test to demonstrate how sets work in removing redundancy by keeping only one copy of each thing. Is it very very redundant in having more than one copy of some words?"

Pseudocode

- Create Map with String Key and Integer Value
- Loop over each `word` in text
- If Map contains (`word`)
 - Increment count Value
 - Else `put (word)` with Value 1
- Print Map when done

Map	
Key <String>	Value <Integer>
Pretend	1
that	1
this	1

Can use a Map to count word occurrences in a body of text

Text from which to identify unique words

"Pretend that this string was loaded from a web page. We won't go to all **that** trouble here. This string contains multiple words. And multiple copies of multiple words. And multiple words with multiple copies. It is to be used as a test to demonstrate how sets work in removing redundancy by keeping only one copy of each thing. Is it very very redundant in having more than one copy of some words?"

Pseudocode

- Create Map with String Key and Integer Value
- Loop over each `word` in text
- If Map contains (`word`)
 - Increment count Value
 - Else put (`word`) with Value 1
- Print Map when done

Map	
Key <String>	Value <Integer>
Pretend	1
that	2
this	1
...	

UniqueWordCounts.java: Use Map to count word occurrences in a body of text

```
9 public class UniqueWordsCounts {  
10    public static void main(String[] args) {  
11        String page = "Pretend that this string was loaded from a web pc  
12        String[] allWords = page.split("[ .,?!]+");  
13        // Declare new Map to hold count of each word  
14        Map<String, Integer> wordCounts = new TreeMap<String, Integer>();  
15        // Loop over all the words split out of the string, adding to map  
16        for (String s: allWords) {  
17            String word = s.toLowerCase();  
18            // Check to see if we have seen this word before, update word count  
19            if (wordCounts.containsKey(word)) ← Check if word seen previously  
20                // Have seen this word before, increment the count  
21                wordCounts.put(word, wordCounts.get(word)+1);  
22            }  
23            else {  
24                // Have not seen this word before, add the new word (Java)  
25                wordCounts.put(word, 1);  
26            }  
27        }  
28        // Print word counts  
29        System.out.println(wordCounts);  
      }
```

Large amount of text simulates webpage

Split into words (aka tokens)

Java has Map based on Trees

Implements Map interface

String Key, Integer Value

Loop over all words

Update word counts

We have seen this word before, increments Value for this Key

Have not seen this word before, put() into Map with a value of 1 for word Key

Printing Map calls toString()

Maps can also contain Objects such as a List as their Value

- Track position where each word appears (first word is at index 0)
- Word may appear in multiple positions (e.g., 7th and 41st index)
- Need a way to track many items for each word (word is Key in Map)
- Use Map with a List as the Value instead of Object representation of a primitive type (e.g., Integer)
- Map will hold many Lists, one List for each Key
- Here each List element is Integer, represents index where word found

Map	
Key<String>	Value <List <Integer>>
Pretend	head
that	head
this	head
...	

Values as objects is a powerful concept indeed!

0 → \

1 → 15 → \

2 → 18 → \

UniqueWordPositions.java: Maps can also contain Objects such as a List as their Value

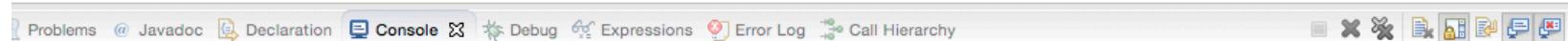
```
9 public class UniqueWordsPositions {  
10    public static void main(String[] args) {  
11        String page = "Pretend that this string was loaded from a web page. We wo  
12        String[] allWords = page.split("[ ,.,?!]+");  
13        // Declare new Map, each entry in the Map is a List that will hold the ind  
14        Map<String, List<Integer>> wordPositions = new TreeMap<String, List<Integer>>;  
15        // Loop over all the words split out of the string, adding their positions  
16        for (int i=0; i<allWords.length; i++) {  
17            String word = allWords[i].toLowerCase();  
18            // Check to see if we have seen this word before, update wordCounts ap  
19            if (wordPositions.containsKey(word)) {  
20                // Now each item in the Map is a List of Integers, add the positio  
21                wordPositions.get(word).add(i);  
22            }  
23            else {  
24                // Add the new word with a new list containing just this position  
25                List<Integer> positions = new ArrayList<Integer>();  
26                positions.add(i);  
27                wordPositions.put(word, positions);  
28            }  
29        }  
30        System.out.println(wordPositions);  
31    }  
32 }
```

Create Map with String as Key and List of Integers as Value

Loop over all words

Update word positions

- If Map has this word as a Key then add() position where word found to List
- get() returns Value which is a List here
- Create a new List if we haven't seen this word before
- add() word to new List
- Then put(word, List) into Map



```
a=[7, 41], all=[14], and=[23, 29], as=[40], be=[38], by=[51], contains=[20], copies=[2]
```

Agenda

1. Set ADT
2. Map ADT
-  3. Reading from file/keyboard
4. Search

UniqueWordPositionsFile.java: Read words from a file instead of hard-coded String

```
12 public class UniqueWordsPositionsFile {  
13     /**  
14      * Collects all the lines from a file into a single string  
15      */  
16     private static String loadFileToString(String filename) throws Exception {  
17         BufferedReader in = new BufferedReader(new FileReader(filename));  
18         String str = "", line;  
19         while ((line = in.readLine()) != null) str += line;  
20         in.close();  
21         return str;  
22     }  
23     Don't forget to close file  
24     public static void main(String[] args) throws Exception {  
25         String page = loadFileToString("inputs/text.txt");  
26         String[] allWords = page.split("[ ,.?]+");  
27  
28         // Declare new Map, each entry in the Map is a List that will hold the index w  
29         Map<String, List<Integer>> wordPositions = new TreeMap<String, List<Integer>>();  
30  
31         // Loop over all the words split out of the string, adding their positions in  
32         for (int i=0; i<allWords.length; i++) {  
33             String word = allWords[i].toLowerCase();  
34  
35             // Check to see if we have seen this word before, update wordCounts approp  
36             if (wordPositions.containsKey(word)) {  
37                 // Now each item in the Map is a List of Integers, add the position of  
38                 wordPositions.get(word).add(i);  
39             }  
39             else {  
40             }  
40         }  
40     }  
40 }
```

- NOTE: Throws exception
- What would happen if file not found?
- Here would pass exception to caller (may end execution)

Append each line from file onto String str

BufferedReader can read from a file on disk

- Load String page from a file
- Rest of the code is the same as UniqueWordsPosition.java

A scanner can be used to read input from keyboard

```
1 import java.util.Scanner;  
2  
3 public class ScannerTest {  
4  
5     public static void main(String[] args) {  
6         Scanner in = new Scanner(System.in);  
7         String line;  
8         int i;  
9         //scanners read from the keyboard  
10        //they can parse input for different types  
11        System.out.println("Enter String");  
12        line = in.nextLine();  
13        System.out.println("Got String: " + line);  
14        //now try reading an integer  
15        System.out.println("Enter integer");  
16        i = in.nextInt();  
17        System.out.println("Got int: " + i);  
18    }  
19 }  
20 }
```

Declare *Scanner* to read from keyboard

Parses input to match assigned type (e.g., read input as a String with *nextLine()*)

Parse input as an integer with *nextInt()*

The screenshot shows the Java IDE interface with the following tabs: Problems, Javadoc, Declaration, Console, Debug, Expressions, Error Log, and Call Hierarchy. The Console tab is active, displaying the following output:

```
<terminated> ScannerTest [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_112.jdk/Contents/Home/bin/java (Jan 2, 2018,  
Enter String  
test  
Got String: test  
Enter integer  
7  
Got int: 7
```

Agenda

1. Set ADT
2. Map ADT
3. Reading from file/keyboard
4. Search

Search.java: Make different data structures to help answer questions

Shakespeare works

Hamlet

Julius Caesar

King Lear

Macbeth

Midsummer

Othello

Romeo & Juliet

Tempest

Read

Key <String> filename	Value Map<<String>,<Integer>>	
word	count	
hamlet.txt	forbear	1
	the	1,150
	...	
juliusCaesar.txt	the	606

Key <String> filename	Value <Integer>
number words	
hamlet.txt	32,831
juliusCaesar.txt	21,183

numFiles: # of files word is in

Key <String> word	Value <Integer>
number files	
forbear	3
forsooth	3
the	8

file2WordCounts

- Use filename as Key
- Store how many times each word appears in file
- Map of Maps!

numWords

- Map filename to number of words in file

totalCounts: How many total times word appears

Key <String> word	Value <Integer>
total count	
forbear	6
forsooth	5
the	5,716

Demo: Search.java uses Scanner and data structures to answer questions

Type a word to see how many times it appears in each file

- Love
- Forbear
- Forsooth
- Audience suggestion

n to get n most common words

- Try top 10 words with # 10, then # 100
- Try bottom 10 words with # -10, then # -100

Can restrict to just a single file with # n (e.g., # 10 hamlet.txt)

Search multiple words, does an AND

Play around on your own

