Eric Roberts
CSCI 121

Handout #48
November 9, 2018

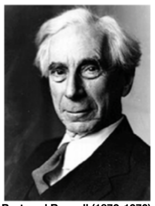
Sets

Sets

Eric Roberts
CSCI 121
November 9, 2018

David Hilbert and Bertrand Russell

- In 1900, the eminent German mathematician David Hilbert set out a series of problems for mathematicians to solve in the 20th century.
- Hilbert believed that mathematical theorems were either true or false. Mathematicians just needed to prove which was which.
- One of the first results to challenge Hilbert's view was Bertrand Russell's discovery that all of set theory was founded on assumptions that were impossible to define.


Bertrand Russell (1872-1970)

Russell's Paradox

Consider the set **B**, which is the set of all sets that do not contain themselves as an element. Does the set **B** contain itself?

Sets in Mathematics

- A ***set*** is an unordered collection of distinct values.

digits = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }
evens = { 0, 2, 4, 6, 8 }
odds = { 1, 3, 5, 7, 9 }
primes = { 2, 3, 5, 7 }
squares = { 0, 1, 4, 9 }

colors = { *red, yellow, green, cyan, blue, magenta* }
primary = { *red, green, blue* }
secondary = { *yellow, cyan, magenta* }

$\mathbf{R} = \{ x \mid x \text{ is a real number} \}$
$\mathbf{Z} = \{ x \mid x \text{ is an integer} \}$
$\mathbf{N} = \{ x \mid x \text{ is an integer and } x \geq 0 \}$

- The set with no elements is called the ***empty set*** (∅).

Set Operations

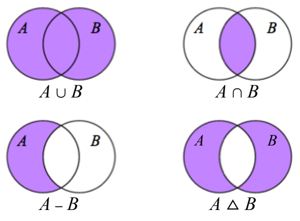
- The fundamental set operation is ***membership*** (∈).

3 ∈ **primes**	3 ∉ **evens**
red ∈ **primary**	*red* ∉ **secondary**
–1 ∈ **Z**	–1 ∉ **N**

- The ***union*** of two sets A and B ($A \cup B$) consists of all elements in either A or B or both.
- The ***intersection*** of A and B ($A \cap B$) consists of all elements in both A or B.
- The ***set difference*** of A and B ($A - B$) consists of all elements in A but not in B.
- The ***symmetric set difference*** of A and B ($A \triangle B$) consists of all elements in A but not in B.

Venn Diagrams

- A ***Venn diagram*** is a graphical representation of a set in that indicates common elements as overlapping areas.
- The following Venn diagrams illustrate the effect of the union, intersection, and the two set-difference operators:



Exercise: Set Operations

Suppose that you have the following sets:

digits = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }
evens = { 0, 2, 4, 6, 8 }
odds = { 1, 3, 5, 7, 9 }
primes = { 2, 3, 5, 7 }
squares = { 0, 1, 4, 9 }

What is the value of each of the following expressions:

a) **evens** ∪ **squares**
b) **odds** ∩ **squares**
c) **squares** ∩ **primes**
d) **primes** – **evens**
e) **odds** △ **squares**

Exercise: Set Operations

Suppose that you have the following sets:

digits = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }
evens = { 0, 2, 4, 6, 8 }
odds = { 1, 3, 5, 7, 9 }
primes = { 2, 3, 5, 7 }
squares = { 0, 1, 4, 9 }

Given only these sets, can you produce the set { 1, 2, 9 }?

Set Relations

- Sets A and B are ***equal*** ($A = B$) if they have the same elements.
- Set A is a ***subset*** of B ($A \subseteq B$) if all elements in A are also in B.
- Set A is a ***proper subset*** of B ($A \subset B$) A is a subset of B but the two sets are not equal.
- If $A = B$, then the circles for A and B in a Venn diagram are the same.
- If $A \subseteq B$, then the circle for A in the Venn diagram lies entirely within (or possibly coincident with) the circle for B.

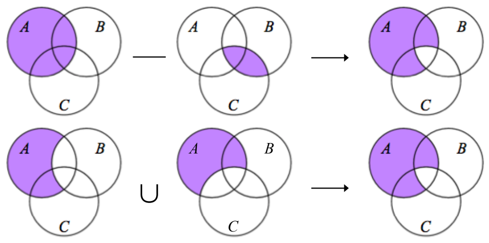
Fundamental Set Identities

$S \cup S \equiv S$ $S \cap S \equiv S$	Idempotence
$A \cup (A \cap B) \equiv A$ $A \cap (A \cup B) \equiv A$	Absorption
$A \cup B \equiv B \cup A$ $A \cap B \equiv B \cap A$ $A \,\triangle\, B \equiv B \,\triangle\, A$	Commutative laws
$A \cup (B \cup C) \equiv (A \cup B) \cup C$ $A \cap (B \cap C) \equiv (A \cap B) \cap C$ $A \,\triangle\, (B \,\triangle\, C) \equiv (A \,\triangle\, B) \,\triangle\, C$	Associative laws
$A \cup (B \cap C) \equiv (A \cup B) \cap (A \cup C)$ $A \cap (B \cup C) \equiv (A \cap B) \cup (A \cap C)$ $A \cap (B \,\triangle\, C) \equiv (A \cap B) \,\triangle\, (A \cap C)$	Distributive laws
$A - (B \cap C) \equiv (A - B) \cup (A - C)$ $A - (B \cup C) \equiv (A - B) \cap (A - C)$	De Morgan's laws

Venn Diagrams as Informal Proofs

- You can also use Venn diagrams to justify set identities. Suppose, for example, that you wanted to prove

$$A - (B \cap C) \equiv (A - B) \cup (A - C)$$



Implementing Sets

- Modern set libraries typically adopt one of two strategies for implementing sets:
  - ***Hash tables***. Sets implemented as hash tables are extremely efficient, offering average $O(1)$ performance for adding a new element or testing for membership. The primary disadvantage is that hash tables make it more costly to iterate through the elements in sorted order.
  - ***Binary search trees***. Sets implemented using a binary search trees (BST) representation offer $O(\log N)$ performance on the fundamental operations, but make it relatively inexpensive to perform a sorted iteration.
- Different language choose different models for sets:
  - Python uses hashed sets.
  - C++ uses the BST model.
  - Java gives clients a choice of models.

Initial Versions Should Be Simple

Premature optimization is the root of all evil.
—Don Knuth

- When you are developing an implementation of a public interface, it is best to begin with the simplest possible code that satisfies the requirements of the interface.
- This approach has several advantages:
  - You can get the package out to clients much more quickly.
  - Simple implementations are much easier to get right.
  - You often won't have any idea what optimizations are needed until you have actual data from clients of that interface. In terms of overall efficiency, some optimizations are much more important than others.

A Dictionary-Based Implementation

- As is so often the case, the easy way to implement a `Set` class is to build it out of data structures that you already have. In this case, the simplest strategy is to build `Set` on top of Python's `dict` class.
- The code for a dictionary-based definition of `Set` that supports methods like `add`, `contains`, and `union` appears on the next several slides.
- The methods that you need from Python's `dict` class are:
  - The ability to create a new key-value pair (assignment to index)
  - The ability to remove a key (`remove` method)
  - The ability to check whether a key exists (`in` operator)
  - The ability to iterate through the keys (`for-in` statement)

Implementation of a `Set` Class

```
# File: Set.py

class Set:

    """Implements a Set class using a dictionary."""

    def __init__(self):
        """Initializes an empty dictionary."""
        self._dict = { }

    def __str__(self):
        """Converts a set to a string."""
        result = ""
        for key in sorted(self._dict):
            if result != "":
                result += ", "
            result += str(key)
        return "{" + result + "}"
```

Implementation of a `Set` Class

```
    def add(self, element):
        """Adds the element to this set."""
        self._dict[element] = True

    def remove(self, element):
        """Removes the element from this set."""
        self._dict.remove(element)

    def clear(self):
        """Removes all elements from this set."""
        self._dict.clear()

    def len(self):
        """Returns the size of this set."""
        return len(self._dict)

    def contains(self, element):
        """Returns True if element is in this set."""
        return element in self._dict
```

Implementation of a `Set` Class

```
    def union(self, other):
        """Returns the union of self and other."""
        newset = Set()
        for key in self._dict:
            newset.add(key)
        for key in other._dict:
            newset.add(key)
        return newset

    def intersection(self, other):
        """Returns the intersection of self and other."""
        newset = Set()
        for key in self._dict:
            if key in other._dict:
                newset.add(key)
        return newset
```

Exercise: Set Implementation

Add the code to implement `issubset` and `difference`.

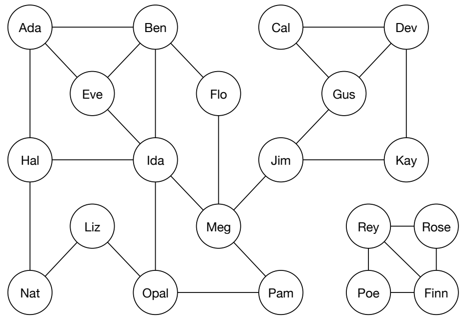
Why Use Sets?

- Sets are useful in programming for several reasons:
  - ***Sets come up naturally in many situations***. Many real-world applications involve unordered collections of unique elements, for which sets are the natural model.
  - ***Sets have a well-established foundation in mathematics***. If you can frame your application in terms of sets, you can rely on various mathematical properties that apply to sets, such as the set identities.
  - ***Sets can make it easier to reason about your programs***. One of the advantages of mathematical abstraction is that using it often makes it easy to think clearly and rigorously about what your program does.
  - ***Many important algorithms are described in terms of sets.*** If you look at web sites that describe some of the most important algorithms in computer science, many of them base those descriptions in terms of set operations.

Application: Tracing Friendship Graphs

- As an example of an application in which sets and algorithms involving sets both appear is the graph of Facebook friends.
- Each Facebook user has a collection of friends. This data structure is most easily modeled as a set because order is not important and duplicates do not arise.
- Over 99% of all Facebook users are connected into one giant network, but there are still isolated pockets in which a group of users have friends from only within a smaller universe.
- One important algorithm for use on such graphs is called ***breadth-first search***, which proceeds outward from a starting node to generate new sets ordered by their length of the friendship path connecting them to the starting node.

A Fanciful Facebook Friend Graph



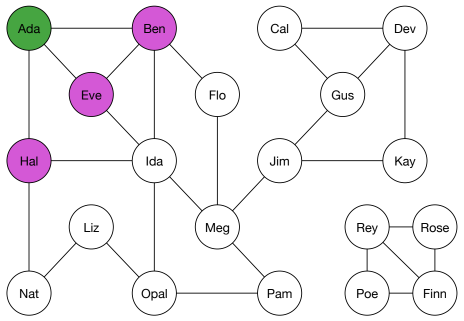
Implementing the Friend Graph

```
FRIEND_GRAPH = {
    "Ada":  [ "Ben", "Eve", "Hal" ],
    "Ben":  [ "Ada", "Eve", "Flo", "Ida" ],
    "Cal":  [ "Dev", "Gus" ],
    "Dev":  [ "Cal", "Gus", "Kay" ],
    "Eve":  [ "Ada", "Ben", "Ida" ],
    "Flo":  [ "Ben", "Meg" ],
    "Gus":  [ "Cal", "Dev", "Jim" ],
    "Hal":  [ "Ada", "Ida", "Nat" ],
    "Ida":  [ "Ben", "Eve", "Hal", "Meg", "Opal" ],
    "Jim":  [ "Gus", "Kay", "Meg" ],
    "Kay":  [ "Dev", "Jim" ],
    "Liz":  [ "Nat", "Opal" ],
    "Meg":  [ "Flo", "Ida", "Jim", "Pam" ],
    "Nat":  [ "Hal", "Liz" ],
    "Opal": [ "Liz", "Meg", "Pam" ],
    "Pam":  [ "Meg", "Opal" ],
    "Rey":  [ "Rose", "Finn", "Poe" ],
    "Rose": [ "Rey", "Poe" ],
    "Poe":  [ "Rey", "Finn" ],
    "Finn": [ "Rey", "Rose", "Poe" ]
```

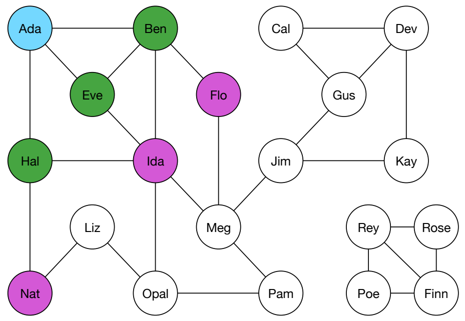
Implementing Breadth-First Search

```
def breadthFirstSearch(g, start):
    """Returns the set reachable from start."""
    frontier = { start }
    reachable = set()
    while len(frontier) > 0:
        expansion = set()
        for name in frontier:
            for p in g[name]:
                if p not in reachable and p not in frontier:
                    expansion.add(p)
        reachable = reachable | frontier
        frontier = expansion
    return reachable
```

Depth-First Search



Depth-First Search


Ada
Ben
Cal
Dev
Eve
Flo
Gus
Hal
Ida
Jim
Kay
Liz
Meg
Rey
Rose
Nat
Opal
Pam
Poe
Finn