Algorithms in Systems Engineering
ISE 172

Lecture 2

Dr. Ted Ralphs

References for Today's Lecture

- Required reading
  - Chapter 1
- References
  - CLRS Chapter 10

What is a Data Structure?

- Computers operate on tables of numbers (the *data*).
- Within the context of solving a given problem, this data has structure.
- *Data structures* are schemes for storing and manipulating data that allow us to more easily see the structure of the data.
- Data structures allow us to perform certain operations on the data more easily.
- The data structure that is most appropriate depends on how the algorithm needs to manipulate the data.
- Commonly used data structures can be made available through the implementation of new *abstract data types*

Examples We'll Consider

- Lists
  - Stacks: Add and delete items in a LIFO way
  - Queues: Add and delete items in a FIFO way
  - Deques: Add and delete items from both ends
  - Priority Queues: Maintain list so that highest priority item is always in front.
- Dictionaries: Look up data by keyword
- Trees: Store data in a hierarchical fashion
- Graphs: Track connections between data elements
- Strings: Data is text that we want to search
- Images: Data is a picture that we want to manipulate

Importance of Data Structures

- Specifying an algorithm completely includes specifying the data structures to be used (sometimes this is the hardest part).
- It is possible for the same basic algorithm to have several different implementations with different data structures.
- Which data structure is best depends on what operations have to be performed on the data.
- Final Example: Conway's Game of Life

Abstract Data Types

- A *data type* is a set of data values and a set of operations that can be performed on those values.
- Data types are a mechanism by which Python and other object-oriented languages allow programmers to define and implement *data structures*.
- The most common data types you have probably encountered so far are the standard built-in data types, such as

- What are the operations we perform on these data types?

Classes in Python

- In Python, *classes* are used to build new data types.
- A class is composed of
  - data attributes, and
  - methods.
- The *data attributes* are the values to be stored and/or operated on.
- The *methods* are the operations to be performed on these values.
- There are also *initializers* and other special methods that control the behavior of the class.

The Interface

- The *interface* defines the way in which *clients* can actually use the data type.
- In object-oriented languages like C++ and Java, the interface consists of the public members of the class.
- The private members of the class, along with function implementations constitute the *implementation*.
- The distinction allows changing the implementation without changing the client program.
- Python does not have this distinction between public and private, but we'll still try to clearly separate the interface from the implementation.
- If necessary, attributes that are meant to be "private" can be given names that are affixed with an "_".

Example: Statistical Data Set

- Suppose we want a new data type for inputting a list of numbers and calculating certain summary statistics on them?
- You can imagine the numbers as being homework grades, for example.
- What are the values to be stored?
- What operations might we want to perform?

Source: Data Structures and Algorithms Using Python and C++ by David Reed and John Zelle

ADT: Statistical Data Set

```
# stats.py

def get_scores():
    """Get scores interactively from the user
    post: returns a list of numbers obtained from the user"""

def min(nums):
    """ find the minimum
    pre: nums is a list of numbers and len(nums) > 0
    post: returns smallest number in nums """

def max(nums):
    """ find the maximum
    pre: nums is a list of numbers and len(nums) > 0
    post: returns largest number in nums """
...
```

Using Python Modules

```
>>> import stats
>>> x = [1, 2, 3]
>>> stats.max(x)
3
>>> stats.min(x)
1
```

Using Classes

```
# Dataset.py
class DataSet:
""" Dataset is a class for computing descriptive statistics
for a set of numbers """

    def __int__(self):
        """post: self is an empty dataset"""

    def add(self, x):
        """add x to the dataset
        pre: x is a list of numbers
        post: x is added to the dataset"""

    def min(self):
        """ find the minimum
        pre: size of self is > 0
        post: returns smallest number in self """
...
```

Using Python Modules

```
>>> from Dataset import Dataset
>>> d = Dataset()
>>> d.add([1, 2, 3])
>>> d.max()
3
>>> d.min()
1
```

Two Implementations of Dataset

- It appears rather straightforward to implement `Dataset`.
- We maintain the list of numbers as a Python list and calculate the statistics on the fly as one would expect.
- Is this efficient?
- It depends...
- A second implementation would be to re-calculate the summary statistics each time new items are added.
- What are the tradeoffs of these two implenentations?

Another Example: A Date Class

- Storing and manipulating dates and times is actually a quite challenging task on a computer.
- What might we want to be able to do with dates?
- What are the challenges?

A Final Example: The Python List Class

- The Python language itself has built-in data structures that programmers can use to store the data of their programs.
- Perhaps the most important built-in data structure is the `list`.
- We will talk about the list class in detail in a coming lecture.
- What sorts of operations does a list class have to support?
- How would you think the list class is implemented and why?