IE 172 Laboratory 1: Algorithm Analysis and Python Tools

Dr. T.K. Ralphs

Due February 11, 2016

1 Laboratory Description and Procedures

1.1 Learning Objectives

1. Understand the use and purpose of classes in Python.
2. Understand how to perform an empirical analysis of an algorithm.
3. Understand how to use the graphics packages `matplotlib` and `pygame` in Python.

1.2 Key Words

1. Object-oriented programming
2. Algorithm Analysis
3. Running Time
4. Pygame
5. Matplotlib

1.3 The Game of Life

The main purpose of this lab is to analyze an implementation of Conway's game of life that has been provided to you. In the next lab, we will do a basic refactoring using more efficient data structures. Conway's game of life is a simulation of the growth of a population that follows several very simple rules. The population is represented by cells represented by locations in a two-dimensional grid (ideally of infinite size, but this is typically not possible in practical implementations). At each time step in the game, the cell at each grid location is either alive or dead. When transitioning from one time step to the next, the following changes in the population occur (see `http://en.wikipedia.org/wiki/Conway's_Game_of_Life`):

- Any live cell with fewer than two live neighbors dies due to under-population.
- Any live cell with two or three live neighbors lives on to the next generation.
- Any live cell with more than three live neighbors dies from overcrowding.
- Any dead cell with exactly three live neighbors becomes a live cell by reproduction.

In this way, the game evolves from one generation to the next. This can easily be visualized graphically, creating interesting visual patterns. However, implementing this efficiently is not actually very easy. In this laboratory, you will learn why this is so challenging in practice.

1.4 Installing the software

Create a new project `Lab1` in your workspace and add the file `life.py` to it. You will also need to install the Python package `pygame`. We'll discuss how to do that in the lab period. After that, you should be able to execute the game as `life(m, n)`, where `m` and `n` are the board dimensions.

1.5 Theoretical Analysis

The algorithm for the game is conceptually simple for a small and finite game board. We can store the current state of the population in a two-dimensional matrix. The challenge is how to efficiently update this data structure from one generation to the next. Let's analyze what the theoretical running time of the algorithm is. Let us consider the size of the input to be the number of cells, i.e., the number of locations in the grid that we start with. Ignoring constant factors, how many "steps" should it to take to apply the rules from Section 1.3 to update the grid from one generation to another? Does the number of steps depend on the input (the beginning state of the game)? See problem 1 below for the formal statement of these questions.

1.6 Empirical Analysis

Now let's take a look at the provided implementation (`life.py`). In this implementation, the initial state of the game is generated randomly, subject to a given initial population density. Run the game for a fixed number of time steps for different sizes of board and graph the result. What function does the running time look like? Try to guess the running time function and compare it to your empirical observations. Be sure to take care of constant factors so that the running times are on the same scale, we scale the theoretical running times are on the same scale, as I showed you in class. See problem 1 below for a formal statement of these questions.

1.7 Object-orientated Programming versus Functional Programming

The code you've been given is written in a *functional* programming style. Such a style is appropriate when you have functions that may operate on many types of data and you perhaps do not even know ahead of time what the data will look like.

Object-oriented code is code in which you have a fixed idea of what the *data* are and the goal is to build a structure for storing that data and manipulating it in certain ways. You may or may not know exactly what you want to do with the data over time or how you want to implement the various methods for manipulating the data. Another more descriptive name for object-oriented programming could be *data-oriented* programming.

In the case of Conway's Game of Life, an object-oriented style seems more appropriate than a function-oriented one because it is the state of the game board that we are storing and manipulating. There are a number of ways of storing this state and performing the required operations, but the data needs to be stored is fixed, as is what we ultimately want to do with it. One of your assignments will be to convert the current function-oriented implementation to an object-oriented one.

2 Laboratory Assignments

2.1 Programming and Analysis (40 points)

1. (10 points) Produce a graph of the running time of the initial implementation for different sizes of boards. The "size" of the board is the number of cells. Square boards are good

enough for testing. You can put the code for producing the graph into `life.py` itself or into a separate file.

2. (10 points) Use this data to conjecture a running time function for the step of updating the board that fits the empirical data and plot your conjectured running time on the same graph.
3. (10 points) Does the empirical running time you observe make sense? Do you think this implementation is as efficient as it could be? Why or why not? How does the running time compare to your the theoretical running time you guessed before doing the empirical experiments?
4. (10 points) Convert the implementation from a function-oriented implementation to an object-oriented one. To do this, introduce a class that contains the state data and then store that data within the class rather than storing it externally and passing it to each function as an argument.
5. (5 points) Create a doc string for the class you introduce.

2.2 Follow-up Questions (20 points)

1. (10 points) Problem 3, Page 79
2. (10 points) Problem 4, Page 79