1 Administrative Topics

- Did everyone get their proj 6 grades?

2 Grammars

For formal languages, grammars define how we create words, or strings, that are legal in that language. That means, formal languages have alphabets and rules about legal strings that can be generated. We use a grammar to take a short string (or a single symbol) and turn it into a long string.

3 Lindenmayer System

We are going to focus on grammars developed by a man named Aristid Lindenmayer (1925-1989), who developed his grammar to model the growth of plants. A grammar of the type developed by Lindenmayer is called a Lindenmayer-system, or L-system.

An L-System has

1. An alphabet. This is a set of symbols, which we will represent with single characters
2. A set of rules, each of which explains how a single symbol can be replaced by one or more symbols.

3. A start symbol or string. This is also called an axiom or initiator.

This week we will be studying systems that have only one rule.

Here is an example:

- My alphabet contains A and B
- My start symbol is A
- My rule is $A \rightarrow AB$

Note that we have a rule about rules: only ONE symbol on the left-hand side.

To create a string in the language defined by this grammar, I begin with the start string and iterate through the rules. The number of times I iterate through the rules determines which string I generate – the more iterations, the longer the string.

Let’s generate a string with one application of the rule in this system:

start: A

A → AB

The result is the string AB

What about with two applications of the rules?

1. string: A. Apply rule ($A \rightarrow AB$)
2. string: AB Apply rule ($A \rightarrow AB$)
3. string: ABB

Thus, to create a string, I need the grammar and the number of times to apply the rules.
Let’s talk more about applying the rules.

Suppose this is our system

alphabet: A,B
rules: A → AB
start: AA

Notice that there are two A’s in the start string? We “simultaneously” apply the rule to all instances of A in the start string, i.e.

AA becomes ABAB.

To show the state of the string after each iteration, I will write it like this:

0: AA
1: ABAB
2: ABBABB

Another way to think of this is that we loop through the symbols in the start string, and apply the rule to each A.

Here is another example:

- alphabet: A, B
- rule: A → BA
- base: BAAB

We apply the first rule by looping through the string. We can also draw it like a tree. e.g.

0: BAAB
   //\   
1: BBABAB
      //\  
2: BBBABBB

Let’s step it up a bit. Design an L-system which will produce the string ”ABBBBBA” after two iterations.
4 Implementing an L-System String Generator

Let’s get the computer to generate strings for us. To do this, we need to store an L-system. We do this by storing them in a list. The first element in the list is start, or base, string. The second element of the list is the rule, which is a list itself (with the left-hand side of the rule (a string) as the first element and the right-hand side of the rule (another string) as the second). Notice that we do not need to store the alphabet explicitly because we rely on the production rules to produce strings for us.

Our goal is to write a function that takes the L-system and a number of iterations as input and returns a string as output. We will write that function (genString) and a few helper functions.
# module to support L-systems, which are represented as lists
# 0: base <str>
# 1: rule: [symbol (l.h.s. of rule) <str>,
# replacement (r.h.s. of rule) <str>]

def getBase(lsyl):
    return lsyl[0]

def getSymbol(lsyl):
    return lsyl[1][0]

def getReplacement(lsyl):
    return lsyl[1][1]

# input: an L-system and the number of iterations to apply
# output: the string generated by the L-system
def genString(lsyl, iters):
    string = getBase(lsyl)
    for i in range(iters):
        string = string.replace(getSymbol(lsyl), getReplacement(lsyl))
    return string

# for module testing
if __name__ == '__main__':
    # base, symbol on l.h.s. of rule, r.h.s. of rule, num iters
    lsyl = ['A', ['A', 'AB']]
    while True:
        input = raw_input('Enter the # of iterations or q to quit: ')
        if input.lower() == 'q':
            break
        print genString(lsyl, int(input))