Mark Eatough

CSIS 2430 9:00 Class

Programming Project 5

Recursion Iteration Contrast Program

Assignment objective:

Compare/contrast Recursive and Iterative methods. Implement recursive and iterative Fibonacci and report which one performs better. Also, implement a recursive modular exponentiation algorithm re-implement the problems solved in the Modular Exponentiation assignment. Write a paragraph with supporting data for each method (hint, you should have a total of 4 paragraphs) comparing/contrasting your results.

What Worked?:

I timed all of my methods inside of a method that called each method to work with four different sets of parameters. The timing methods returned the total time converted to a floating point number. I then had a comparison method which divided the total time of the longer method by the total time of the shorter method. With these smaller sets of data it is a little hard to wrap my head around how much faster one function is versus another. So this allowed me too see it more easily, (for example I could see that function A was 5 times faster than function B).

What did not work?:

Within one of my methods I needed to use an else if statement, the python compiler was complaining of a syntax error. Upon further research I found that the keyword elif has the same meaning as else if in other languages which I found interesting. I also found that my iterative modular exponentiation method only worked when the number was raised to an even power. This error partially came from following the example out of the book. Because the book used an even example and the last digit of the binary number was a 0, nothing was done in the first case, when in fact I needed to multiply (x) by (b) if the last digit in the binary number was a one. At first I thought my recursive function had a bug so I was trying to figure out what was wrong with it, but it was my iterative method which had issues the whole time.

Comments:

My iterative modular exponentiation function calls a function to convert the exponent to a binary number, and then figures out the modulus of the large number by successively squaring the base, taking its modulus, and then based on whether or not the digit in the binary number is a 0 or a 1 multiplying our final answer variable by our the modulus of our base, and then taking the modulus of our number. After timing both functions I found that the recursive method is about 1.2 times faster than the iterative method.

My recursive modular exponentiation function returns one if the exponent is zero. It returns the modulus of the method squared, using the base, half the exponent, and the modulus as the parameters if the exponent is even. If the exponent is odd it returns the method squared, using the base, the floor as half the exponent, and the modulus as the parameters, then we take the modulus of the modulus times the base, and then take the modulus of that whole number. After timing both functions I found that the recursive method is about 1.2 times faster than the iterative method.

My iterative Fibonacci method return 0 is the number equals zero, otherwise it sets a variable x to zero, a variable y to one, and for a variable (i) equal to 1 and less than the parameter passed to the function we set z equal to x plus y, x equal to y, and y equal to z. At the end of the for loop we return y. After timing both functions I found that the iterative method is about 250 times faster than the recursive method.

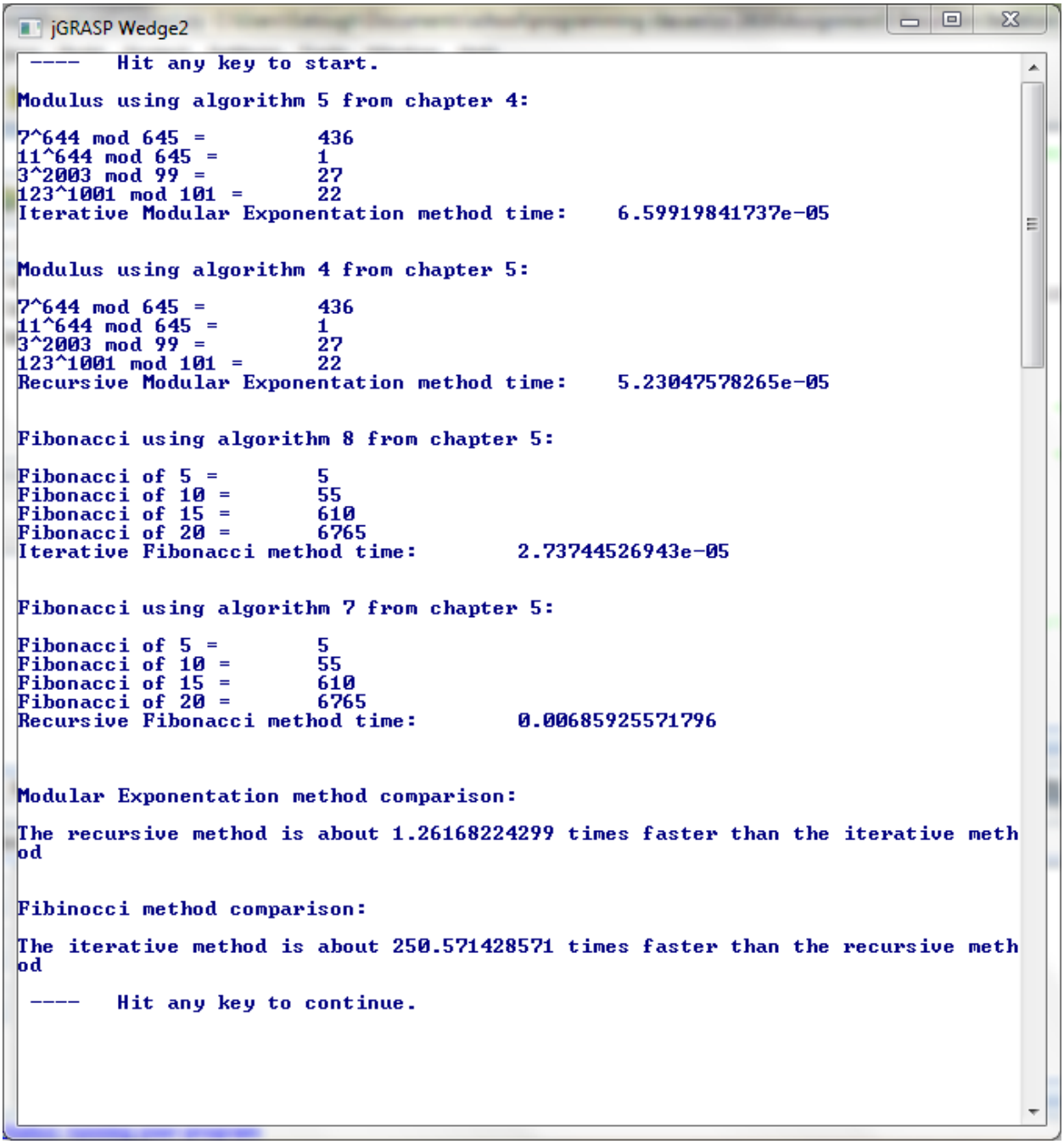
The recursive Fibonacci method returns the parameter if the parameter n is equal to zero or 1. Otherwise the method returns the method using the n minus one as the parameter plus the method using n minus two as the parameter. After timing both functions I found that the iterative method is about 250 times faster than the recursive method.

```
'''
************************************************************
 * Discrete Structures
 * Recursion Iteration Contrast Program
 * Programmer: Mark Eatough
* Course: CSIS 2430
 * Created September 29, 2013

 ************************************************************
'''
import time

#algorithm 1 from chapter 4 in book, used to find binary
representation of my exponents
def makeBinary(number):
   q = number
   b = 2
   binary = ""
   while(q > 0):
      a = q%b
     q = q/b
      binary = str(a) + binary
   return binary

#algorithm 5 from chapter 4 in book, iterative modular
exponentiation algorithm
#this one is partially wrong?
def iterativeModularExponentation(number, exponent, modulus):
   x = 1
   binary = makeBinary(exponent)
   i = len(binary)-1
   if(binary[i] == '1'):
      x = x*number
      x = x% modulus
   while (i > 0):
      number*=number
      number = number % modulus
      if(binary[i-1] == '1'):
         x*=number
         x = x % modulus
      i-=1
   return x

#method to square something, used in recursive modular
exponentation
def squared(x):
   return x*x
```

```
#algorithm 4 from chapter 5 in book, recursive modular
exponentiation equation
#this one is all right?
def recursiveModularExponentation(number, exponent, modulus):
   if(exponent == 0):
      return 1
   elif(exponent % 2 == 0):
      return squared(recursiveModularExponentation(number,
exponent/2, modulus)) % modulus
   else:
      return (squared(recursiveModularExponentation(number,
exponent/2, modulus))%modulus*number%modulus) % modulus
#algorithm 7 from chapter 5 in book, recursive fibonacci
algorithm
def recursiveFibonacci(number):
   if(number  <= 1):
      return number
   else:
      return recursiveFibonacci(number - 1) +
recursiveFibonacci(number - 2)
#algorithm 8 from chapter 5 in book, iterative fibonacci
algorithm
def iterativeFibonacci(number):
   if(number == 0):
      return number
   else:
      x = 0
      y = 1
      for i in range(1, number):
         z = x + y
         x = y
         y = z
      return y

def timeIterativeMod():
   itModStart = time.clock()
   itMod1 = iterativeModularExponentation(7,644, 645)
   itMod2 = iterativeModularExponentation(11,644, 645)
   itMod3 = iterativeModularExponentation(3,2003, 99)
   itMod4 = iterativeModularExponentation(123,1001, 101)
   #calculate time
   itModTime = time.clock() - itModStart
   #print modulus
   print "\nModulus using algorithm 5 from chapter 4:\n"
   print "7^644 mod 645 = \t", itMod1
```

```
    print "11^644 mod 645 = \t", itMod2
    print "3^2003 mod 99 = \t", itMod3
    print "123^1001 mod 101 = \t", itMod4
    #print time it took
    print "Iterative Modular Exponentation method time: \t",
itModTime
    return float(itModTime)

def timeRecursiveMod():
    recModStart = time.clock()
    recMod1 = recursiveModularExponentation(7,644, 645)
    recMod2 = recursiveModularExponentation(11,644, 645)
    recMod3 = recursiveModularExponentation(3,2003, 99)
    recMod4 = recursiveModularExponentation(123,1001, 101)
    #calculate time
    recModTime = time.clock() - recModStart
    #print modulus
    print "\n\nModulus using algorithm 4 from chapter 5:\n"
    print "7^644 mod 645 = \t", recMod1
    print "11^644 mod 645 = \t", recMod2
    print "3^2003 mod 99 = \t", recMod3
    print "123^1001 mod 101 = \t", recMod4
    #print time it took
    print "Recursive Modular Exponentation method time: \t",
recModTime
    return float(recModTime)

def timeIterativeFib():
    itFibStart = time.clock()
    itFib1 = iterativeFibonacci(5)
    itFib2 = iterativeFibonacci(10)
    itFib3 = iterativeFibonacci(15)
    itFib4 = iterativeFibonacci(20)
    #calculate time
    itFibTime = time.clock() - itFibStart
    #print fibinacci
    print "\n\nFibonacci using algorithm 8 from chapter 5:\n"
    print "Fibonacci of 5 = \t", itFib1
    print "Fibonacci of 10 = \t", itFib2
    print "Fibonacci of 15 = \t", itFib3
    print "Fibonacci of 20 = \t", itFib4
    #print time it took
    print "Iterative Fibonacci method time: \t", itFibTime
    return float(itFibTime)

def timeRecursiveFib():
    recFibStart = time.clock()
```

```
    recFib1 = recursiveFibonacci(5)
    recFib2 = recursiveFibonacci(10)
    recFib3 = recursiveFibonacci(15)
    recFib4 = recursiveFibonacci(20)
    #calculate timne
    recFibTime = time.clock() - recFibStart
    #print fibinacci
    print "\n\nFibonacci using algorithm 7 from chapter 5:\n"
    print "Fibonacci of 5 = \t", recFib1
    print "Fibonacci of 10 = \t", recFib2
    print "Fibonacci of 15 = \t", recFib3
    print "Fibonacci of 20 = \t", recFib4
    #print time it took
    print "Recursive Fibonacci method time: \t", recFibTime
    return float(recFibTime)

def timeCompare(it, rec):
    if it < rec:
        print "The iterative method is about", rec/it, "times
faster than the recursive method"
    else:
        print "The recursive method is about", it/rec, "times
faster than the iterative method"

itMod = timeIterativeMod()
recMod = timeRecursiveMod()
itFib = timeIterativeFib()
recFib = timeRecursiveFib()

print "\n\n\nModular Exponentation method comparison:\n"
timeCompare(itMod, recMod)
print "\n\nFibinocci method comparison:\n"
timeCompare(itFib, recFib)
```


jGRASP Wedge2
---- Hit any key to start.
Modulus using algorithm 5 from chapter 4:
7^644 mod 645 = 436
11^644 mod 645 = 1
3^2003 mod 99 = 27
123^1001 mod 101 = 22
Iterative Modular Exponentation method time: 6.59919841737e-05
Modulus using algorithm 4 from chapter 5:
7^644 mod 645 = 436
11^644 mod 645 = 1
3^2003 mod 99 = 27
123^1001 mod 101 = 22
Recursive Modular Exponentation method time: 5.23047578265e-05
Fibonacci using algorithm 8 from chapter 5:
Fibonacci of 5 = 5
Fibonacci of 10 = 55
Fibonacci of 15 = 610
Fibonacci of 20 = 6765
Iterative Fibonacci method time: 2.73744526943e-05
Fibonacci using algorithm 7 from chapter 5:
Fibonacci of 5 = 5
Fibonacci of 10 = 55
Fibonacci of 15 = 610
Fibonacci of 20 = 6765
Recursive Fibonacci method time: 0.00685925571796
Modular Exponentation method comparison:
The recursive method is about 1.26168224299 times faster than the iterative meth
od
Fibinocci method comparison:
The iterative method is about 250.571428571 times faster than the recursive meth
od
---- Hit any key to continue.