Tribonacci Numbers

September 14, 2012

You will recall that fibonacci numbers are formed by a sequence starting with 0 and 1 where each succeeding number is the sum of the two preceeding numbers; that is, F[n] = F[n-1] + F[n-2] with F[0] = 0 and F[1] = 1. We studied fibonacci numbers in a previous exercise (/2010/07/30/fibonacci-numbers/).

Tribonacci numbers are like fibonacci numbers except that the starting sequence is 0, 0 and 1 and each succeeding number is the sum of the three preceeding numbers; that is, T[n] = T[n-1] + T[n-2] + T[n-3] with T[-1] = 0, T[0] = 0 and T[1] = 1. The powering matrix for tribonacci numbers, used similarly to the powering matrix for fibonacci numbers, is:

$$\begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}$$

The first ten terms of the tribonacci sequence, ignoring the two leading zeros, are 1, 1, 2, 4, 7, 13, 24, 44, 81 and 149 (A000073 (http://oeis.org/A000073)).

Your task is to write two functions that calculate the first *n* terms of the tribonacci sequence by iteration and the *n*th term by matrix powering; you should also calculate the tribonacci constant, which is the limit of the ratio between successive tribonacci numbers as *n* tends to infinity. When you are finished, you are welcome to read (/2012/09/14/tribonacci-numbers/2/) or run (http://programmingpraxis.codepad.org/mxq30iLw) a suggested solution, or to post your own solution or discuss the exercise in the comments below.

Pages: 1 2 (https://programmingpraxis.com/2012/09/14/tribonacci-numbers/2/)

Posted by programmingpraxis
Filed in Exercises
8 Comments »

8 Responses to “Tribonacci Numbers”

1. *Paul* said
   September 14, 2012 at 5:54 PM
   A python version that uses the numpy library for matrices

```
import itertools as IT
import numpy

M = numpy.matrix([[1,1,0], [1, 0 ,1], [1, 0, 0]], dtype=float)

def tribonacci():
    a, b, c = 0, 0, 1
    while 1:
        yield c
        a, b, c = b, c, a + b + c

def tripow(N):
    return int((M ** N)[0,0])

print list(IT.islice(tribonacci(), 10)) #-> [1, 1, 2, 4, 7, 13, 24
print tripow(9)                          # -> 149
x = list(IT.islice(tribonacci(), 1000))
print x[-1] / float(x[-2])               # -> 1.83928675521
```

2. *Paul* said
   September 14, 2012 at 9:18 PM
   Another Python version with the power of a matrix explicitly in the code.

```
import itertools as IT

M = [[1,1,0], [1, 0 ,1], [1, 0, 0]]

def vecvec(a, b):
    return sum(ai*bi for ai, bi in zip(a, b))

def matmul(a, b):
    """ number of columns in a must be equal to number of rows in
    bt = zip(*b)
    return [[vecvec(ai, btj) for btj in bt] for ai in a]

def mpow(a, nn):
    """"matrix a in the power nn
        Build the 1, 2, 4, 8 ... powers of a and find binary of nn
        Finally multiply the required powers of a
    """
    powers = [a]
    binary = []
    while nn:
        binary.append(nn & 1)
        nn /= 2
        if nn:
            powers.append(matmul(powers[-1], powers[-1]))
    mats = (powi for powi, bi in zip(powers, binary) if bi)
    return reduce(lambda a, b: matmul(a, b), mats)

def tribonacci():
    a, b, c = 0, 0, 1
    while 1:
        yield c
        a, b, c = b, c, a + b + c

def tripow(N):
    return mpow(M, N)[0][0]

print list(IT.islice(tribonacci(), 20)) #-> [1, 1, 2, 4, 7, 13, 24
print tripow(9)                         # -> 149
x = list(IT.islice(tribonacci(), 1000))
print x[-1] / float(x[-2])              # -> 1.83928675521
print tripow(35) / float(tripow(34))    # -> 1.83928675521
```

3. *Jan Van lent* said

September 15, 2012 at 1:04 PM

Calculation of the limit of the ratio as one of the roots the characteristic polynomial.
This is also the real eigenvalue of the powering matrix.

```
/* Maxima transcript */

(%i40) r : rhs(solve(z^3-z^2-z-1, z)[3]);
                  sqrt(11)   19 1/3            4             1
(%o40)           (-------- + --)    + -------------------- + -
                     3/2     27            sqrt(11)   19 1/3   3
                    3                 9 (-------- + --)
                                            3/2     27
                                           3
(%i41) string(optimize(r));
(%o41)  block([%1],%1:(sqrt(11)/sqrt(3)^3+19/27)^(1/3),%1+4/(9*%1)
(%i42) float(r);
(%o42)                         1.839286755214161
```

4. *Graham* said
September 15, 2012 at 3:10 PM
It’s been a while since I’ve used GNU Octave (Matlab clone), so I thought I’d give it a try. Not particularly clever, but it was good practice to brush up on working with matrices.

```
format long;

function t = tribo(n)
    a = 0; b = 0; t = 1;
    for i = 2:n
        tmpa = a; tmpb = b; tmpt = t;
        a = b; b = t; t = a + b + t;
    endfor
    return;
endfunction

function t = tripow(n)
    T = [1, 1, 0; 1, 0, 1; 1, 0, 0];
    t = (T^n)(1, 1);
    return;
endfunction

function l = lim(n)
    l = tripow(n) / tripow(n - 1);
    return;
endfunction
```

5. *cage* said
September 15, 2012 at 4:25 PM
My implementation in Common lisp

6. *treeowl* said
September 16, 2012 at 8:34 PM
As I mentioned in a recent comment on the fibonacci post, it’s actually impossible to achieve O(log n) performance for arbitrarily large n. The sequence is (to a close approximation) exponential, so the lengths of the binary or decimal values increase linearly. It is thus impossible for any algorithm to run in sublinear time.

7. *Catalin Cristu (@catalin_c)* said
September 18, 2012 at 8:03 AM
Another Python solution:

```python
def gen_tribonacci():
    a, b, c = 0, 0, 1
    while True:
        yield c
        a, b, c = b, c, a + b + c

def nth_tribonacci(n):
    m = matpow([[1, 1, 0],
                [1, 0, 1],
                [1, 0, 0]], n)
    return m[2][0]

def matpow(m, n):
    if n == 1:
        return m
    if n % 2:
        return matmult(m, matpow(matmult(m, m), n / 2))
    return matpow(matmult(m, m), n / 2)

def matmult(m1, m2):
    return [[dot(row, col) for col in zip(*m2)] for row in m1]

def dot(a, b):
    return sum([x * y for x, y in zip(a, b)])

def tribonacci_const(n):
    ts = take(n, gen_tribonacci())
    return float(ts[-1]) / ts[-2]

def take(n, g):
    result = []
    for _ in xrange(n):
        result.append(g.next())
    return result

if __name__ == '__main__':
    print(take(10, gen_tribonacci()))
    print(nth_tribonacci(8))
    print(tribonacci_const(1000))
```

8. *Victor Chavauty* said
September 24, 2012 at 3:19 AM
#include
using namespace std;

int main()
{
unsigned long int Counter;
unsigned long int result = 0;
unsigned long int numa, numb, numc;
numa = 0;
numb = 0;
numc = 1;
unsigned long int number;
cout << "Qual o numero que desejas conhecer da sequencia tribonacci?" << endl <> number;
cout << endl << endl;
for(Counter = 0; Counter < number – 1; Counter++)
{

```
result = numa + numb + numc;
numa = numb;
numb = numc;
numc = result;
}
cout << "O Valor Tribonacci do numero " << number << " e igual a: " << result;
return 0;
}
```