Fibonacci and the The Perfect Microbes

The perfect microbe is very-young for 1 second, young for the next, and old for the next and subsequent seconds. Each old microbe produces a new microbe every second, i.e. 2 seconds after the creation of a microbe, the microbe produces a new microbe and another new microbe for subsequent seconds.

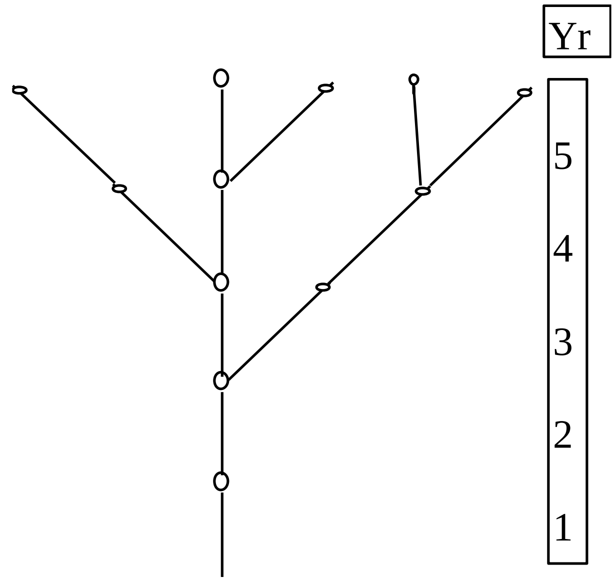
Assume we start with just one microbe.

During Second#	#Microbes
1	1
2	1
3	2
4	3
5	5
6	8
...	...

Tree Growing Branches

Each branch grows during the 1st year and the end of each subsequent year, it grows a new branch.

During Yr	1	tree has	1	branch
“	2	“	1	“
“	3	“	2	“
“	4	“	3	“
“	5	“	5	“
“	6	“	8	“
“	7	“	13	“



Counting Ways up Stairs

One can either take one step or two steps at at time.

Stairs:

#steps	1	2	3	4	n
ways	step 1	{step 1, step 2} or {step 2}	{step 1, step 2, step 3} or {step 1, step 3} or {step 2, step 3}	{step 1, then 3 steps left} or {step 2, then 2 steps left}	{step 1, then n-1 steps left} or {step 2, then n-2 steps left}
total	1	2	3	ways(3)+ways(2)=5	ways(n-1)+ways(n-2)

Fibonacci Sequence

Leonardo Fibonacci (‘son of Bonacci’) is a nickname for Leonardo Pisano Bigollo.

fib(0)	fib(1)	fib(2)	fib(3)	fib(4)	fib(5)	fib(6)	fib(7)	...	fib(12)
0	1	1	2	3	5	8	13	...	144

Inductive/Recursive Definition

```
fib(0) = 0
fib(1) = 1
fib(k+2) = fib(k+1) + fib(k), if k ≥ 0
```

The following Java function is based on this inductive/recursive definition of the function, `fib`,

```
int fib(int k)
    {
        if ( k == 0 )
            return 0;
        else if ( k == 1 )
            return 1;
        else
            return fib(k - 2) + fib(k - 1);
    } // fib
```

More efficient and iterative Java functions can be given as:

```java
int fib1(int k)
{   // iterative solution

    int j, p, c, n;

    p = 0;
    c = 1;
    j = 1;
    while ( j < k )
    {
        n = p + c;
        p = c;
        c = n;
        j = j+1;
    }
    return c;

} // fib1
```

```java
int  fib2(int  k)
{   // iterative solution

    int c = 0;
    int n = 1;
    for (int j = 1; j < k ; j = j+1)
    {
      n = n + c;
      c = n - c;
    }
    return n;

} // fib2
```

Golden Ratio, ϕ

line L = |------------|------------------------|
1 ϕ

$\phi \; is\ the\ Golden\ Ratio$

$\equiv \; \frac{1}{\phi} = \frac{\phi}{1+\phi}$

$\equiv \phi^2 - \phi - 1 = 0$

ϕ is the positive root of $x^2 - x - 1$,

$\phi = \frac{1+\sqrt{5}}{2}$

i.e. $\phi \approx 1.618$.

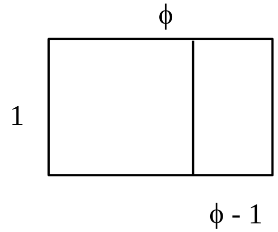
Other root of $x^2 - x - 1$ is

$\hat{\phi} = \frac{1-\sqrt{5}}{2}$ (= -0.618)

Note: roots of $a * x^2 + b * x + c \;\; are \; \frac{-b \pm \sqrt{b^2 - 4*a*c}}{2*a}$.

Golden Rectangle

A rectangle is said to be 'Golden' if it sides are in Golden Ratio.
Consider a rectangle with width 1 and length ϕ. If the unit square is removed, the rectangle left is still a Golden Rectangle



$$\frac{\phi}{1} = \frac{1}{\phi - 1}$$

$$\equiv \quad \phi^2 - \phi - 1 = 0.$$

Lemma:

$\phi^n = \phi^{n-1} + \phi^{n-2}$ (also $\hat{\phi}^n = \hat{\phi}^{n-1} + \hat{\phi}^{n-2}$), for $n \geq 2$. [Note: $\phi^0 = 1$]

Pf:

Since $\phi^2 - \phi - 1 = 0$,
we have, $\phi^2 = \phi + 1$
tf. $(n \geq 2)$

$$\begin{aligned} \phi^n &= \phi^2 \phi^{n-2} \\ &= (\phi+1)\phi^{n-2} \\ &= \phi^{n-1} + \phi^{n-2} \end{aligned}$$

Theorem. $fib(n) = \frac{\phi^n - \hat{\phi}^n}{\sqrt{5}}$

Pf: (by induction)
Base Cases:

n= 0

fib(0) = 0 = $\frac{\phi^0 - \hat{\phi}^0}{\sqrt{5}}$

n=1

fib(1) = 1

and

$$\frac{\phi^1 - \hat{\phi}^1}{\sqrt{5}} = \frac{1 + \sqrt{5} - 1 + \sqrt{5}}{2\sqrt{5}} = 1$$

Induction Step: (n>2)

$$fib(n) \quad = fib(n-1) + fib(n-2)$$

$$= \frac{\phi^{n-1} - \widehat{\phi}^{n-1}}{\sqrt{5}} + \frac{\phi^{n-2} - \widehat{\phi}^{n-2}}{\sqrt{5}}$$

$$= \frac{\phi^{n-1} - \widehat{\phi}^{n-1} - (\phi^{n-2} - \widehat{\phi}^{n-2})}{\sqrt{5}}$$

$$= \frac{\phi^{n} - \widehat{\phi}^{n}}{\sqrt{5}} \quad \{ \text{ by Lemma } \}$$

Corollary:

The notation [x] can be used for ‘x rounded to nearest integer’

Then $fib(n) = \left[\frac{\phi^n}{\sqrt{5}}\right]$

The number, ϕ^n can be calculated using a (log n) algorithm.

(See Matt Parker on Numberphile: https://www.youtube.com/watch?v=PeUbRXnbmms)

A Java function based on this result can be given as:

```
long fib3(int k)
    {
        double phi, s5, result;
        Exponent_Power ep;

        ep = new Exponent_Power();
        s5 = ep.newton_sqrt(5.0);
        phi = (1 + s5)/2;
        result = ep.fast_exp(phi, k) / s5;
        return Math.round(result);
    } // fib3
```

This function make use of the log(n) function,

`double fast_exp(double a, int b)`,

in the class, `Exponent_Power`, which also contains the function,

`double newton_sqrt(double n)`.

Alternatively, the functions,

`double pow(double a, double b)`

and

double `sqrt(double a)`

from the Java Library class, `Math`, could be used.

Other Properties of the Fibonacci function:

- $\lim_{n\to\infty} \frac{fib(n+1)}{fib(n)} = \phi, the\ Golden\ Ratio.$

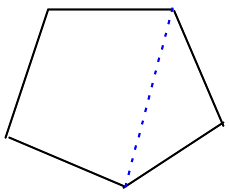
- $fib(n+1) * fib(n-1) = (fib(n))^2 + (-1)^n$

  e.g. fib(7)*fib(5) = (fib(6)^2 +(-1)^6

  13 * 5 = 8^2 + 1

  65 = 64 + 1

- A regular pentagon with sides 1, has a 'diagonal' of length ϕ. |Diag| = 2 Cos $\frac{\pi}{5}$ = ϕ



- fib(2*n+1) = $(fib(n))^2 + (fib(n+1))^2$ i.e. fib(2*n+1) is sum of two squares.

  fib(2*n) = fib(n)*(2*fib(n+1) - fib(n))

Fast Fibonacci function

Based of the mathematical result

- fib(2*n+1) = $(\text{fib}(n))^2 + (\text{fib}(n+1))^2$
- fib(2*n) = fib(n)*(2*fib(n+1) - fib(n))

we have the following (log n) recursive function for fibonacci

```
int fast_fib(int k)
{
 if ( k == 0 )
    return 0;
 else if ( k == 1 || k == 2 )
        return 1;
      else
      {
         int k2 = k/2;
         if ( k%2 == 0 )
              return  fast_fib(k2)*(2*fast_fib(k2+1)-fast_fib(k2));
         else
              return fast_fib(k2)*fast_fib(k2) + fast_fib(k2+1)*fast_fib(k2+1);
      }
 } // fast_fib
```

Example: Calculate fast_fib(13)

From table above we have

fib(0)	fib(1)	fib(2)	fib(3)	fib(4)	fib(5)	fib(6)	fib(7)	...
0	1	1	2	3	5	8	13	...

$fib(13)$

$= (fib(6))^2 + (fib(7))^2$

$= 8^2 + 13^2$

$= 64 + 169$

$= 233$

Optimising `fast_fib`

B can calculate the next fibonacci number by

$$fib(k+1) \; = \; [\varphi * fib(k)]$$

as

$$\lim_{n\to\infty} \frac{fib(n+1)}{fib(n)} = \varphi, the\ Golden\ Ratio.$$

e.g. fib(5) = [φ*fib(4)] = [φ*3] = [1.618*3] = [4.854]= 5,
we can optimise the function, `fast_fib`, to `fast_fib1`

We are assume, the constant, `phi`, has been defined in the class.

```
int fast_fib1(int k)
{
     if ( k  == 0 )
          return 0;
    else if ( k == 1 || k == 2 )
         return 1;
    else if ( k == 3)
         return 2;
    else
         {   int k2 = k/2;
             int f1 = fast_fib1(k2);
             int f2 = (int)Math.round(phi*f1);
                       // casting is needed as Math.round returns a double
             //f2 = fast_fib1(k2+1);
             if ( k%2 == 0 )
                 return f1*(2*f2 - f1);
                     //return  fast_fib1(k2)*(2*fast_fib1(k2+1)-fast_fib1(k2)) ;

             else
                 return f1*f1 + f2*f2;
                    //return fast_fib1(k2)*fast_fib1(k2) +
                                        fast_fib1(k2+1)*fast_fib1(k2+1) ;
         }
}  // fast_fib1
```

```java
public class Fib_Calc
{
    double phi;

    Fib_Calc()
    {
        phi = (1+Math.sqrt(5.0))/2;
    } // Fib_Calc


    int fib(int k)
    {
        if ( k == 0 )
            return 0;
        else if ( k == 1 )
            return 1;
        else
            return fib(k - 2) + fib(k - 1);
    } // fib
```

```
int fib1(int k)
{   // iterative solution

    int j, p, c, n;

    p = 0;
    c = 1;
    j = 1;
    while ( j < k )
    {
        n = p + c;
        p = c;
        c = n;
        j = j+1;
    }
    return c;
} // fib1
```

```
int  fib2(int  k)
{   // iterative solution
    int c = 0;
    int n = 1;
    for (int j = 1; j < k ; j = j+1)
    {
        n = n + c;
        c = n - c;
    }
    return n;
} // fib2
```

```
long fib3(int k)
{
    double phi, s5, result;
    Exponent_Power ep;

    ep = new Exponent_Power();
    s5 = ep.newton_sqrt(5.0);
    phi = (1 + s5)/2;
    result = ep.fast_exp(phi, k) / s5;
    return Math.round(result);
} // fib3
```

```
int fast_fib(int k)
{
 if ( k == 0 )
    return 0;
 else if ( k == 1 || k == 2 )
        return 1;
      else
      {
         int k2 = k/2;
         if ( k%2 == 0 )
              return  fast_fib(k2)*(2*fast_fib(k2+1)-fast_fib(k2));
         else
              return fast_fib(k2)*fast_fib(k2) + fast_fib(k2+1)*fast_fib(k2+1);
      }
 } // fast_fib
```

```
    int fast_fib1(int k)
    {
        if ( k  == 0 )
             return 0;
        else if ( k == 1 || k == 2 )
            return 1;
        else if ( k == 3)
            return 2;
        else
            {   int k2 = k/2;
                int f1 = fast_fib1(k2);
                int f2 = (int)Math.round(phi*f1);
                         // casting is needed as Math.round returns a double
                //f2 = fast_fib1(k2+1);
                if ( k%2 == 0 )
                    return f1*(2*f2 - f1);
                        //return  fast_fib1(k2)*(2*fast_fib1(k2+1)-fast_fib1(k2)) ;

                else
                    return f1*f1 + f2*f2;
                       //return fast_fib1(k2)*fast_fib1(k2) +
                                           fast_fib1(k2+1)*fast_fib1(k2+1) ;
            }
    }  // fast_fib1

} //  class Fib_Calc
```

Zeckendorf representation

A natural number n can be represented as a sum of Fibonacci numbers where successors in the sequence are not successive Fibonacci numbers.
100 = 89 + 8 + 3
512 = 377 + 89 + 34 + 8 + 3 + 1
Two successive Fibonacci number can be replaced by their sum as a Fibonacci number is the sum of its two previous Fibonacci numbers.
Every natural number *n* has a unique Zeckendorf representation.

Program to find Zeckendorf representation

To find Zeckendorf representation of a number n, find the largest Fibonacci number, m, less than n. Let ms be the Zeckendorf representation of n – m, then m:ms is the Zeckendorf representation of n.
Example: n = 100
Largest Fibonacci number less than 100 = 89.
The Zeckendorf representation of 100 – 89 = 11 is [8,3] and therefore
Zeckendorf representation of 100 is 89:[8,3] = [89,8,3].

Find largest Fibonacci number less than max.

```
int max_fib(int max)
{ // Largest fibonnacci number less than max.
    int p, c, n;

    p = 0;
    c = 1;
    while ( c <= max )
    {
        n = p + c;
        p = c;
        c = n;
    }
    return p;
}
```

Mininum Int Value: -2,147,483,648 (= - 2^{31})

Maximum Int Value: 2,147,483,647 (= $2^{31} - 1$)

Exercise: Find largest Fibonacci number of type, **int**, less than (or equal to) 2,147,483,647 without using integer type, **long**.
Fib(46) = 1,836,311,903 but Fib(47) = 2,971,215,073 which is bigger than $2^{31} - 1$

Zeckendorf Representation

```
String fib_rep(int n)
{
    int m, mxf;
    String result;

    result = "";
    m = n;
    mxf = max_fib(m);
    while ( m != mxf )
    {
        result = result + mxf + "+";
        m = m - mxf;
        mxf = max_fib(m);
    }
    result = result + mxf;
    return result;
}
```

Converting Miles to Kilometres

1 Mile ≈ 1.61 km and Golden Ratio (φ) ≈ 1.62

$\therefore \quad n\, miles \;\approx\; (\varphi * n)\; km$

$\therefore \quad fib(n)\, miles \;\approx\; (\varphi * fib(n)\,)\; km$

But,

$$\frac{fib(n+1)}{fib(n)} \approx \varphi \qquad \therefore \quad fib(n+1) \approx \varphi * fib(n)$$

$\therefore \quad fib(n)\, miles \;\approx\; fib(n+1)\; km$

e.g $8\, miles \;\approx\; 13\, km$

We can use the Zechendorf representation of a number, $n,$ to convert n miles when n is not a fibonacci number. Let $n = 50$:

$50 = 34 + 13 + 3$ (Zechendorf representation)

Replacing each fib(n) with fib(n+1) we get the conversion

$81 = 55 + 21 + 5$

$\therefore \; 50\, miles \;\approx\; 81\, km.$

Similarly, for the reverse conversion of km to $miles$, replace $fib(n)$ with $fib(n-1)$.

e.g.

$81\, km = 55 + 21 + 5$ converts to $34 + 13 + 3 = 50\, miles$

Alternatively, since $\varphi \approx \frac{8}{5} = 1.6$ and using $n\ miles \approx (\varphi * n)\ km$ i.e.

$$n\ miles \approx \left(\frac{8}{5} * n\right) km$$

we get

$50\ miles \approx (\frac{8}{5} * 50)\ km$ i.e.

$50\ miles \approx 80\ km$

Similarly,

$$n\ km \approx \left(\frac{5}{8} * n\right) miles$$