Fibonacci Sequence and Matrices

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(n+2) = fib(n+1) + fib(n), if n $\geq$ 0

‘Fibonacci Matrices’

Let

$$F = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$$

then

$$F^2 = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} * \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix}$$

then

$$F^3 = \begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix} * \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 2 \\ 2 & 3 \end{bmatrix}$$

then

$$F^4 = \begin{bmatrix} 1 & 2 \\ 2 & 3 \end{bmatrix} * \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 2 & 3 \\ 3 & 5 \end{bmatrix} = \begin{bmatrix} fib(3) & fib(4) \\ fib(4) & fib(5) \end{bmatrix}$$

By induction it can be shown that

$$F^n = \begin{bmatrix} fib(n-1) & fib(n) \\ fib(n) & fib(n+1) \end{bmatrix}$$

Fast Fibonacci Program (Log n)

For matrix

$$F = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$$

we have

$$F^n = \begin{bmatrix} fib(n-1) & fib(n) \\ fib(n) & fib(n+1) \end{bmatrix}$$

We can use a (log n) algorithm for exponents to find F^n .

Note: For a matrix, M, $M^0 = Id$ where *Id* is the Identity matrix.

It is assumed that M is a square matrix as if M is an $r \times c$ $(r \neq c)$ matrix then $M * M$ is not defined.

Matrix_Math Class

The Matrix_Math class describes matrix objects with properties such as multiplying by another matrix and equality of Matrices.

Two constructor methods are given:

- `Matrix_Math(int r, int c)`
  create an all zero matrix of size, $r \times c$.

- `Matrix_Math(int[][] arr_2d)`
  create a matrix from a two dimensional array, `arr_2d`.

```
class Matrix_Math
{
    int rows;           // #rows
    int cols;           // #columns
    int[][] item;       // 2-d array

    Matrix_Math(int r, int c)
    {// create r x c matrix of 0's
      rows = r;
      cols = c;
      item = new int[r][c];
    } // Matrix_Math

    Matrix_Math(int[][] arr_2d)
    { // create matrix based on 2d array,
        rows = arr_2d.length;
        cols = arr_2d[0].length;
        item = new int[rows][cols];
        for (int i = 0; i < rows; i=i+1)
            for (int j = 0; j < cols; j=j+1)
                item[i][j] = arr_2d[i][j];
    } // Matrix_Math
```

Matrix Identity

```
Matrix_Math identity(int n)
{
    Matrix_Math Id = new Matrix_Math(n,n);
    for (int i = 0; i < n; i=i+1)
        Id.item[i][i] = 1;
    return Id;
} // identity
```

Matrix Copy

The function, `copy`, returns a copy of the matrix object.

```
Matrix_Math copy_mat()
{
    Matrix_Math result = new Matrix_Math(rows, cols);
    for (int i = 0; i < rows; i=i+1)
        for (int j = 0; j < cols; j=j+1)
            result.item[i][j] = item[i][j];
    return result;
    // result.eq(this)
} // copy_mat
```

Matrix Multiplication

The numbers of rows in B must the same as the the columns in the matrix object.

```
Matrix_Math times(Matrix_Math B)
{ assert( cols == B.rows );
    Matrix_Math C = new Matrix_Math(rows, B.cols);
    for (int i = 0; i < C.rows; i=i+1)
        for (int j = 0; j < C.cols; j=j+1)
            for (int k = 0; k < cols; k=k+1)
                C.item[i][j] = C.item[i][j] + item[i][k]*B.item[k][j];
    return C;
} // times
```

Recall, for **double** a and **int** b, the (log n) algorithm for finding a^b ,

```
double fast_exp(double a, int b)
 { assert ( b >= 0 );

     double x = a;
     int k = b;
     double r = 1.0;

     while (k != 0)
         if ( k%2 == 0 )
         {
             x = x*x;
             k = k/2;
         }
         else
         {
             r = r*x;
             k = k-1;
         }
     return r;
 //Post: r = a^b
 } // fast_exp
```

Matrix Exponent, M^n

```
Matrix_Math f_exp_mat(int n)
{ // rows = cols  (square matrix) && n >= 0
        Matrix_Math result = new Matrix_Math(rows,cols);
        Matrix_Math mm = new Matrix_Math(rows,cols);
        if ( n == 0 )
            result = identity(rows);
        else if ( n == 1 )
                result = copy_mat();
             else
```

```
            {  // n > 1
               mm = copy_mat();
               int k = n;
               result = identity(rows);
               while ( k != 0 )
                   if ( k%2 == 0 )
                   {
                       mm = mm.times(mm);
                       k = k/2;
                   }
                   else
                   {
                       result = result.times(mm);
                       k = k-1;
                   }
            }
        return result;
        // result = this^n
} // f_exp_mat
```

Calculation Fibonacci numbers using matrices.

```
int fib_mat(int n)
{ // n >= 0

    int result;
    Matrix_Math fmat;
    int [][] init = {{0,1},{1,1}};
    fmat = new Matrix_Math(init);
    if (n == 0)
        result = 0;
    else if ( n == 1 )
            result = 1;
         else
            result = fmat.f_exp_mat(n-1).item[1][1];

    return result;
} // fib_mat
```

Matrix Equality

Java allows exiting from the middle of a loop using the '`return`' statement.

Using the `return` ***statement.***

```
boolean eq(Matrix_Math B)
    {// Matrix equality
        assert( (B.rows == rows) && (B.cols == cols) );
        for (int i = 0; i < rows; i=i+1)
            for (int j = 0; j < cols; j=j+1)
                if ( B.item[i][j] != item[i][j] ) return false;
        return true;
    } //eq
```

Not using the `return` ***statement***

In some styles of programming, the Java statement, `return`, is regarded as a particular use of the GOTO statement.
(See Wikipedia: http://en.wikipedia.org/wiki/Goto)

> *"Other academics took the completely opposite viewpoint and argued that even instructions like* `break` *and* `return` *from the middle of loops are bad practice as they are not needed in the Böhm-Jacopini result, and thus advocated that loops should have a single exit point. For instance, Bertrand Meyer wrote in his 2009 textbook that instructions like* `break` *and* `continue` *"are just the old* `goto` *in sheep's clothing"."*

In this particular case of the function, `eq`, it is needed to exit both the inner loop and the outer loop without using a GOTO equivalent.

```
boolean eq(Matrix_Math B)
{// Matrix equality
    assert( (B.rows == rows) && (B.cols == cols) );
    int i, j, r, c ;
    r = rows;
    i = 0;
    while ( i < r )
    {
        j = 0;  c = cols;
        while ( j < c )
            if ( B.item[i][j] ==  item[i][j] )
                j = j+1;
            else
                c = j;
        if ( j == cols )
            i = i+1;
        else
            r = i;
    }
    return ( i == rows );
} // eq
```

In general, programs without GOTOs are easier to prove correct.

Show

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

Proof:

Since

$$F^n = \begin{bmatrix} fib(n-1) & fib(n) \\ fib(n) & fib(n+1) \end{bmatrix}$$

we have

$$F^{2n} = \begin{bmatrix} fib(2n-1) & fib(2n) \\ fib(2n) & fib(2n+1) \end{bmatrix}$$

But

$$F^{2n} = (F^n)^2$$

i.e.

$$\begin{bmatrix} fib(2n-1) & fib(2n) \\ fib(2n) & fib(2n+1) \end{bmatrix} = \begin{bmatrix} fib(n-1) & fib(n) \\ fib(n) & fib(n+1) \end{bmatrix} * \begin{bmatrix} fib(n-1) & fib(n) \\ fib(n) & fib(n+1) \end{bmatrix}$$

$$= \begin{bmatrix} (fib(n-1))^2 + (fib(n))^2 & fib(n-1) * fib(n) + fib(n) * fib(n+1) \\ fib(n) * fib(n-1) + fib(n+1) * fib(n) & (fib(n))^2 + (fib(n+1))^2 \end{bmatrix}$$

tf.

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

and

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

but $fib(n-1) = fib(n+1) - fib(n)$

tf.

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