

プログラム言語 B フィボナッチ数列

2023 年 12 月 1 日

指導教員：武田先生

6321120

横溝 尚也

1 課題 1

次に定義されるフィボナッチ数列 F_i を考える。

$$F_0 = 0, F_1 = 1, F_n = F_{n-1} + F_{n-2} \quad (n \geq 2)$$

この数列を列挙してみると 0, 1, 1, 2, 3, 5, 8, 13, 21, ... となる。

この数列を求めるためのメソッド

```
long fib(int n)
```

を定義に従って再帰的に記述しなさい。

ただし、メソッド呼び出し `fib(i)` の戻り値が F_i に対応する。

プログラムの内容と説明は課題 2 の部分で行う。

2 課題 2

上で定義した `fib` メソッドを用いて、次のプログラムを使って、 $i=50$ までのフィボナッチ数列 F_i を計算しなさい。

```
// 処理前の時刻を取得
```

```
long startTime = System.currentTimeMillis();
```

```
for(int i=0; i<=50; i++){
```

```
System.out.println("fib("+i+")= "+fib(i));
```

```
}
```

```
// 処理後の時刻を取得
```

```
long endTime = System.currentTimeMillis();
```

```
// 処理時間の表示
```

```
System.out.println("処理時間：" + (endTime - startTime) + " msec");
```

2.1 プログラムの内容

プログラム 1 : Fib.java

```
1 public class Fib {
2
3     public static void main(String[] args) {
4         // 処理前の時刻を取得
5         long startTime = System.currentTimeMillis();
6         for(int i=0; i<=50; i++){
7             System.out.println("fib("+i+")= "+fib(i));
8         }
9         // 処理後の時刻を取得
10        long endTime = System.currentTimeMillis();
11        // 処理時間の表示
12        System.out.println("処理時間：" + (endTime - startTime) + " msec");
```

```
13     }
14
15     static long fib(int n){
16         if (n == 0){
17             return 0;
18         }else if(n == 1){
19             return 1;
20         }else{
21             return fib(n-1) + fib(n-2);
22         }
23     }
24 }
```

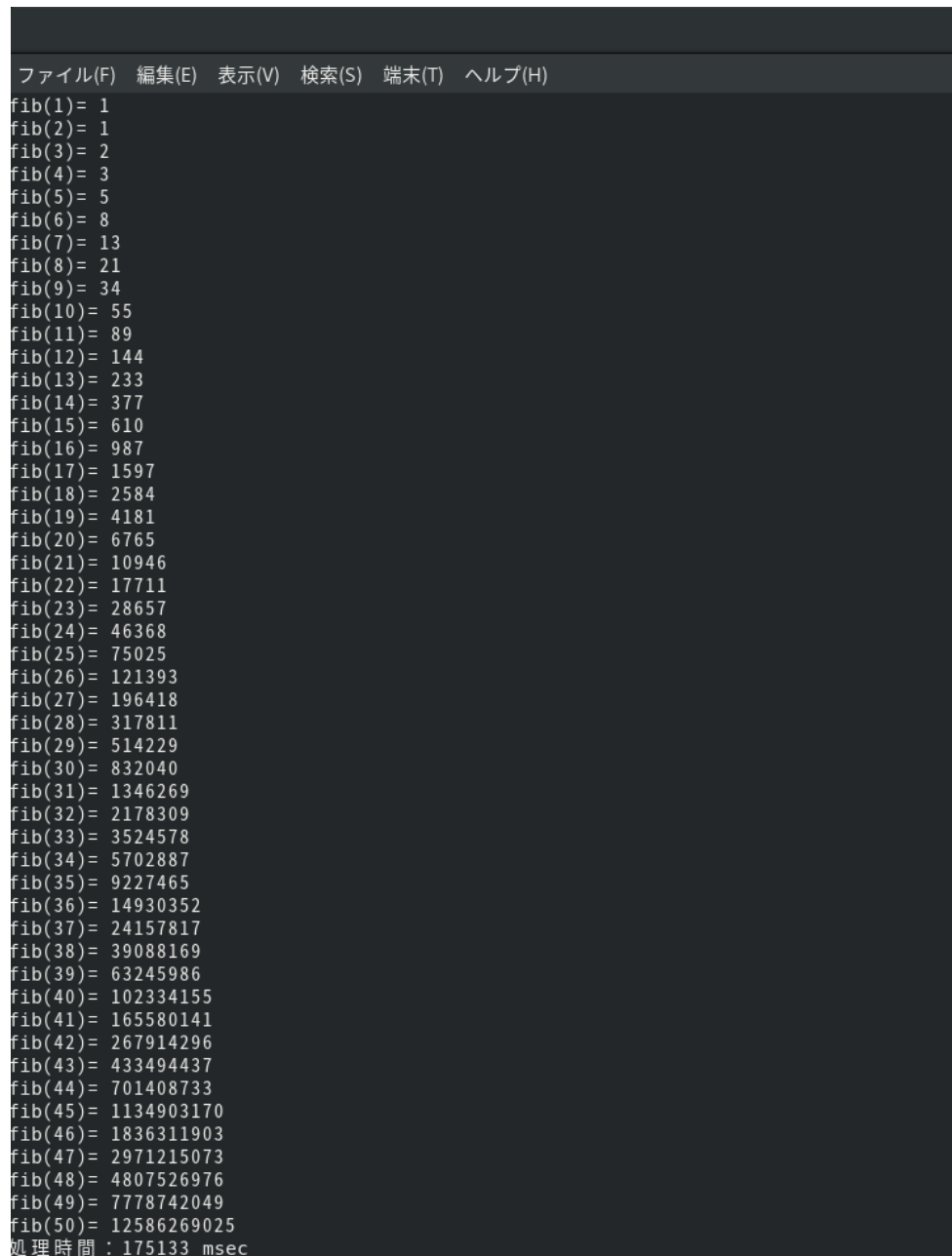
1～13 行目

課題内容で与えられているプログラムを参考にしたため、説明は省略する。

15～23 行目

fib メソッドの記述である。定義通りに、 $F_0 = 0, F_1 = 1, F_n = F_{(n-1)} + F_{(n-2)}$ を if 文を使用して返戻値をそれぞれ定めている。

2.2 実行結果



```
ファイル(F) 編集(E) 表示(V) 検索(S) 端末(T) ヘルプ(H)
fib(1)= 1
fib(2)= 1
fib(3)= 2
fib(4)= 3
fib(5)= 5
fib(6)= 8
fib(7)= 13
fib(8)= 21
fib(9)= 34
fib(10)= 55
fib(11)= 89
fib(12)= 144
fib(13)= 233
fib(14)= 377
fib(15)= 610
fib(16)= 987
fib(17)= 1597
fib(18)= 2584
fib(19)= 4181
fib(20)= 6765
fib(21)= 10946
fib(22)= 17711
fib(23)= 28657
fib(24)= 46368
fib(25)= 75025
fib(26)= 121393
fib(27)= 196418
fib(28)= 317811
fib(29)= 514229
fib(30)= 832040
fib(31)= 1346269
fib(32)= 2178309
fib(33)= 3524578
fib(34)= 5702887
fib(35)= 9227465
fib(36)= 14930352
fib(37)= 24157817
fib(38)= 39088169
fib(39)= 63245986
fib(40)= 102334155
fib(41)= 165580141
fib(42)= 267914296
fib(43)= 433494437
fib(44)= 701408733
fib(45)= 1134903170
fib(46)= 1836311903
fib(47)= 2971215073
fib(48)= 4807526976
fib(49)= 7778742049
fib(50)= 12586269025
処理時間 : 175133 msec
```

図 1 Clock.py の出力結果

上記の画像から、このアルゴリズムではフィボナッチ数列の第 50 項目の計算には約 175 秒もかかってしまうことがわかる。

3 課題 3

再帰的定義に従うと、 $\text{fib}(i)$ を求めるために要する加算の回数は、

$$\text{fib}(i+1) - 1$$

の式で与えられるように見えるが、一般に、この式が成り立つことを数学的帰納法により証明しなさい。

Proof.

(i) $i = 0$ のとき

$$\text{fib}(0+1) - 1 = 0 \text{ より成立}$$

(ii) $i = 1$ のとき

$$\text{fib}(1+1) - 1 = 0 \text{ より成立}$$

(iii) $i = k, i = k + 1$ のときの成立を仮定、すなわち

$$(\text{fib}(k) \text{ の加算回数}) = \text{fib}(k+1) - 1, (\text{fib}(k+1) \text{ の加算回数}) = \text{fib}(k+2) - 2 \text{ とする。}$$

$$\text{このとき、} (\text{fib}(k+2) \text{ の加算回数}) = (\text{fib}(k+1) \text{ の加算回数}) + (\text{fib}(k) \text{ の加算回数}) + 1$$

$$(\because \text{fib}(k+2) = \text{fib}(k+1) + \text{fib}(k))$$

$$\text{仮定より、} (\text{fib}(k+2) \text{ の加算回数}) = (\text{fib}(k+2) - 1) + (\text{fib}(k+1) - 1) + 1$$

$$= \text{fib}(k+3) - 1$$

(i),(ii),(iii) から数学的帰納法より $i \geq 0$ である整数 i に対して $\text{fib}(i+1) - 1$ が成立。

■

4 課題 4

$\text{fib}(i)$ の計算に要する加算の回数を、 i に比例する計算量におさえるような効率の良いフィボナッチ数列を求めるメソッド

`long fib2(int n)`

を記述しなさい。

4.1 プログラムの内容

プログラム 2 : `Fibnew.java`

```
1 public class Fib_new{
2
3     public static void main(String[] args) {
4         // 処理前の時刻を取得
5         long startTime = System.currentTimeMillis();
6         for(int i=0; i<=50; i++){
7             System.out.println("fib("+i+")= "+fib2(i));
8         }
9         // 処理後の時刻を取得
```

```

10     long endTime = System.currentTimeMillis();
11     // 処理時間の表示
12     System.out.println("処理時間：" + (endTime - startTime) + " msec");
13 }
14
15 // メモ化再帰を使用した効率的なフィボナッチ数列の計算メソッド
16 public static long fib2(int n) {
17     long[] memo = new long[n + 1];
18     return fib2_sub(n, memo);
19 }
20
21 private static long fib2_sub(int n, long[] memo) {
22     if (n == 0) {
23         return 0;
24     } else if (n == 1) {
25         return 1;
26     } else if (memo[n] != 0) {
27         // メモに結果があれば再計算せずにその結果を返す
28         return memo[n];
29     } else {
30         //  $F_n = F_{n-1} + F_{n-2}$  の再帰的な計算
31         long result = fib2_sub(n - 1, memo) + fib2_sub(n - 2, memo);
32         // 計算結果をメモに保存
33         memo[n] = result;
34         return result;
35     }
36 }
37 }

```

1～13 行目

説明を省略

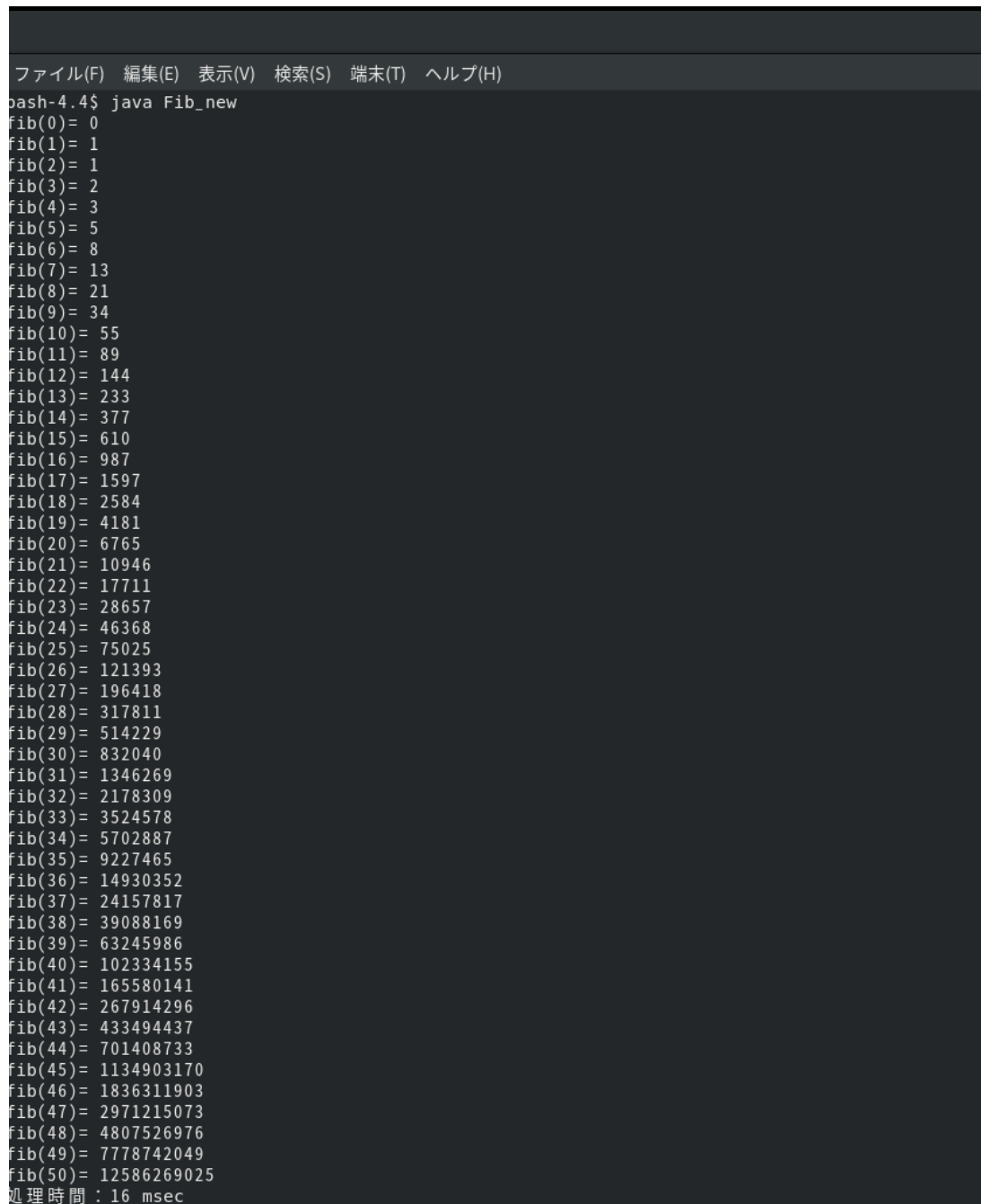
16～19 行目

fib2 メソッドの記述である。課題1とは異なり、fib2_new メソッドで数列の値を計算し、一度計算した項の値は配列 memo に格納し、以後一度計算済みの項に関しては、再度計算を行わず、配列を参照する。

21～36 行目

fib_new メソッドの記述である。課題1と同様に $n=0, n=1$ の処理を場合分けを行って戻り値を定めることに変更はない。それ以外の場合、フィボナッチ数列の定義通りすべての項を計算するのではなく、配列 memo を参照して、一度でも計算済みの項に関しては、配列に格納してある値を返戻地として返す。

4.2 実行結果



```
bash-4.4$ java Fib_new
fib(0)= 0
fib(1)= 1
fib(2)= 1
fib(3)= 2
fib(4)= 3
fib(5)= 5
fib(6)= 8
fib(7)= 13
fib(8)= 21
fib(9)= 34
fib(10)= 55
fib(11)= 89
fib(12)= 144
fib(13)= 233
fib(14)= 377
fib(15)= 610
fib(16)= 987
fib(17)= 1597
fib(18)= 2584
fib(19)= 4181
fib(20)= 6765
fib(21)= 10946
fib(22)= 17711
fib(23)= 28657
fib(24)= 46368
fib(25)= 75025
fib(26)= 121393
fib(27)= 196418
fib(28)= 317811
fib(29)= 514229
fib(30)= 832040
fib(31)= 1346269
fib(32)= 2178309
fib(33)= 3524578
fib(34)= 5702887
fib(35)= 9227465
fib(36)= 14930352
fib(37)= 24157817
fib(38)= 39088169
fib(39)= 63245986
fib(40)= 102334155
fib(41)= 165580141
fib(42)= 267914296
fib(43)= 433494437
fib(44)= 701408733
fib(45)= 1134903170
fib(46)= 1836311903
fib(47)= 2971215073
fib(48)= 4807526976
fib(49)= 7778742049
fib(50)= 12586269025
処理時間: 16 msec
```

図2 Fib_new.py の出力結果

上記の画像から、課題1で記述したすべての項を再帰的に計算するアルゴリズムよりも、課題4のように一度計算済みの項は配列に値を格納し、そこから値を参照することでオーダーが減少し、処理時間が劇的に減少することがわかった。