## 0.4, a and b only

## Yin.Zheng

January 30, 2022

## 1 Code examples

```
def fib_matrix(n):
      if n == 1:
2
          return 1
      if n == 2:
          return 1
5
6
          F1 = [0, 1, 1, 1]
          return naive_matrix_power(F1, n)[0]
9
10
def naive_matrix_power(F1, n):
12
13
      B = [1, 1]
      for _ in range(n-1):
14
          B = matrix_multiply(F1, B)
15
16
          global number_of_matrix_muliply
          number_of_matrix_muliply += 1
17
          # print(B)
18
      return B
19
20
21
def matrix_multiply(F1, B):
23
      a, b, c, d = F1
      x, y = B
24
      global add_time
25
      global multi_time
26
      add_time += 2
27
      multi_time += 4
      return (a*x+b*y, c*x+d*y)
29
32 number_of_matrix_muliply = 0
33 \text{ add\_time} = 0
34 \text{ multi\_time} = 0
n = 8 # input your nth fibonacci number here
grint("the", n, "number of fibonacci number is", fib_matrix(n))
37 print("the number of matrix muliply is", number_of_matrix_muliply)
38 print("the number of addition is performed is", add_time)
39 print("the number of multiplication is performed is", multi_time)
```

Listing 1: This is O(n) version of my fib matrix

```
1
_{2} F1 = [0, 1, 1, 1]
5 def matrix_fib(n):
     if n == 1:
6
          return 1
      if n == 2:
8
9
          return 1
      else:
10
          result = matrix_power(F1, n)
11
          return result[1]
12
13
14
def matrix_power(F1, n):
      if n == 0:
16
17
         return [1, 0, 0, 1]
      elif n == 1:
18
19
          return F1
      else:
20
         B = F1
21
         i = 2
22
23
          while i <= n:
               # repeated square B until n = 2^q > m
24
               B = matrix_multiply_f2(B, B)
25
26
               global global_N
               global_N += 1
27
               i = i*2
28
          # add on the remainder
29
          R = matrix_power(F1, n-i//2)
30
31
          return matrix_multiply_f2(B, R)
32
33
34 def matrix_multiply_f1(F1, B):
35
      a, b, c, d = F1
      x, y = B
36
      return (a*x+b*y, c*x+d*y)
37
38
39
40 def matrix_multiply_f2(A, B): # this function returns matrix A*B
      a, b, c, d = A
41
      x, y, z, w = B
42
     global add_time
43
     global multi_time
44
45
      add_time += 4
      multi_time += 8
46
      return (
47
          a*x + b*z,
48
          a*y + b*w,
49
          c*x + d*z,
50
          c*y + d*w,
51
52
53
54
# 1,1,2,3,5,8,13,21,34,55
56 global_N = 0
57 \text{ add\_time} = 0
```

```
58 multi_time = 0
59 n = 8
60 print("the ", n, "th fib number is ", matrix_fib(n))
61 print("The is O(logn)", global_N)
62 print("the number of addition is performed is", add_time)
63 print("the number of multiplication is performed is", multi_time)
```

Listing 2: This is  $O(\log(n))$  version of my fib with matrix

(a) Q1:Show that two 2 \* 2 matrices can multiplied using 4 additions and 8 multiplications. But how many matrix multiplications does it take to compute Xn?

Answer: In my first version of matrix fib it takes N-1 steps to compute  $X^n$ , Because every time n increases by 1, it is multiplied once more by [0,1,1,1] so the time complexity is O(n)

```
def naive_matrix_power(F1, n):
                                                                                                                                                                                                                                                                                                            Print Output:
                    B = [1, 1]
                                                                                                                                                                                                                                                                                                             the 8 number of fibonacci number is 21
the number of matrix muliply is 7
the number of addition is performed is 14
the number of multiplication is performed is 28
                     for _ in range(n-1):
    B = matrix_multiply(F1, B)
                                        global number_of_matrix_muliply
                                       number_of_matrix_muliply += 1
                                       # print(B)
                     return B
 def matrix_multiply(F1, B):
                    global add_time
global multi_time
                   add_time += 2
multi_time += 4
                      return (a*x+b*y, c*x+d*y)
number_of_matrix_muliply = 0
 add_time = 0
multi_time = 0
print("the", n, "number of fibonacci number is", fib_mat
print("the number of matrix muliply is", number_of_matri
 print("the number of addition is performed is", add_time
 print("the number of multiplication is performed is", multiplication is performed is "print", multiplication is performed in the "print", multiplication is performed in the performed is perfo
```

Figure 1: Screen shot for Q1

(b) Q2:Show that O(logn) matrix multiplications suffice for computing Xn. (Hint:Think about computing X8.)

Answer:In my second version of matrix Fib, it takes it takes log(n) steps to compute  $X^n$  Because in this version matrix Multiplication grows exponentially For example when n=8, we are calculating the 8th fib number. This is what happens in this loop:

```
while i <= n:
    # repeated square B until n = 2^q > m
B = matrix_multiply_f2(B, B)
global global_N
global_N += 1
i = i*2
```

Listing 3: loop code

- When i=2 B became  $B^2$
- when i=4  $B^2$  became  $B^4$
- when i=8  $B^4$  became  $B^8$
- when i=16 jump out of the loop

As you can see the matrix multiply function execute 3 times, 3 = log(8), so O(logn) matrix multiplications suffice for computing  $X^n$ 

```
D ∨ □ 以 □
                           🧽 my_fib_matrix.py U 🔹 🧽 my_fib_v2.py U
OPEN EDITORS 1 UNSAVED
                          CS5800 > my_fib_v2.py > matrix_multiply_f1
 • 👶 my_fib_matrix... U
× → my_fib_v2.py... U

GROUP 2
                                                  R = matrix_power(F1, n-i//2)
                                                  return matrix_multiply_f2(B, R)
                                                                                                                                          Print Output:
    LiveCode - my_fib..
                                                                                                                                          the 8 th fib number is 21
The is O(logn) 3
the number of addition is performed is 16
the number of multiplication is performed is 32
MY_LTCD_SLT
                                    def matrix_multiply_f1(F1, B):
                              34
                                            a, b, c, d = F1
 i binary_search
                                           x, y = B
return (a*x+b*y, c*x+d*y)
 binary_tree
  multipy.py U
my_fib_matrix... U
                                    def matrix_multiply_f2(A, B): # this function returns
    🌪 my_fib_v2.py U
                                           a, b, c, d = A
                                           x, y, z, w = B
global add_time
global multi_time
   etest_2.py
 test.py U
 ■ linked-list-cycle
■ majority-element
                                            add_time += 4
 merge-two-sorted-l.
                                            multi_time += 8
                                                 a*x + b*z,
                                                  a*y + b*w,
                                                  c*x + d*z,
                                                  c*y + d*w,
                              56 global_N = 0
                                     add time = 0
                              58 multi_time = 0
                              59 n = 8
60 print("the ", n, "th fib number is ", matrix_fib(n))
61 print("The is O(logn)", global_N)
62 print("the number of addition is performed is", add_time
63 print("the number of multiplication is performed is", mu
```

Figure 2: Screen shot for Q2