Lecture 2: Fibonacci Search

Victor Gan

September 10, 2013

1 INTRODUCTION

The previous lecture introduced two computational models (Turing Machines and Random Access Machines). Further models are briefly listed here. This lecture demonstrates how to analyse the time complexity of algorithms by using the example of Fibonacci numbers and walking through multiple algorithms to calculate them.

2 OTHER COMPUTATIONAL MODELS

Besides Turing Machines and Random Access Machines, other computational models include parallel computation, quantam models and randomized computation. Randomized computation involves randomness as part of its logic. For example, the Miller-Raben test [1] efficiently tests the primality of a number by running a test on random integers multiple times. These models may be further elaborated on later in the course.

3 FIBONACCI NUMBERS

The Fibonacci Numbers [2] are the integer sequence

$$1, 1, 2, 3, 5, 8, 13...$$

where each subsequent number in the sequence is the sum of the previous two. Mathematically,

$$f_n = f_{n-1} + f_{n-2}$$

This sequence of numbers appears in many biological settings such as fruit sprouts of a pineapple and the arrangement of a pine cone, but it is also of interest due to its applications in the Fibonacci search algorithm and Fibonacci heaps. Two of many algorithms are described here to compute the Fibonacci series.

3.1 Algorithm 1: Exponential Time

The first algorithm is perhaps the most intuitive but leads to unsatisfactory time performance using a RAM model. The algorthim for calculating the nth number in the Fibonacci sequence, where n is a positive integer is:

```
fib(n):
    (if n = 1) return 1
    (if n = 2) return 1
    (if n > 2) return fib(n-1) + fib(n-2)
```

Let us use n = 5 as an example. To calculate f_5, the fifth Fibonacci number, we must first calculate f_4 and f_3 since

$$f_5 = f_4 + f_3$$

Similarly, to calculate f_4, we must first first find f_3 (which relies on f_2 and f_1) and f_2, and to to calculate f_3, we must first first find f_2 and f_1. Hence, f_1 is computed twice, f_2 is computed three times, f_3 is computed twice, f_4 is computed once and f_5 is computed once. Notably, many Fibonacci numbers are computed more than once. To further analyse the time complexity, we need a concrete method of calculating how long this algorithm takes to run. One method is to model the time it takes to find f_n for $n > 2$ as

$$Timetofindf_n = (timetofindf_{n-1}) + (timetofindf_{n-2}) + overhead(n)$$

or

$$T(n) = T(n-1) + T(n-2) + overhead(n) \tag{3.1}$$

A more simpler method is to count the number of additions,

$$numberofadditionsf_n = (numberofadditionsf_{n-1}) + (numberofadditionsf_{n-2}) + 1$$

or

$$A(n) = A(n-1) + A(n-2) + 1 \tag{3.2}$$

In fact, 3.2 is a lower bound of 3.1 assuming $T(1) \geq A(1)$ and $T(2) \geq A(2)$. This can be proven by induction. Then, solving the recurrence relation of 3.2 leads to an exponential time complexity.

3.2 Algorithm 2: Linear Time

Consider a second algorithm that calculates nth number in the Fibonacci sequence.

```
variables:
    last_value
    sec_last_value
    new_value

last_value = 1
sec_last_value = 1
for i = 3 to n do
    new_value = last_value + sec_last_value
    sec_last_value = last_value
    last_value = new_value
end
return new_value
```

This method avoids recalculating Fibonacci numbers by storing the two previous Fibonacci numbers in memory, and by doing so, leads to a linear time complexity.

4 Conclusion

This lecture, in addition to introducing the concept of recursion, recurrence relations and calculating time complexity, shows the importance of algorithm design. Two algorithms that lead to the same result may have significantly different performance characteristics, making evaluation of these algorithms important. A more in-depth evaluation of algorithms in terms of time and space complexity, as well as more sophisticated examples of algorithms that improve on these characteristics, are to be addressed in upcoming lectures.

References

[1] Wikipedia, "Miller-Rabin primality test" http://en.wikipedia.org/wiki/Miller-Rabin_primality_test

[2] Wikipedia, "Fibonacci number" http://en.wikipedia.org/wiki/Fibonacci_number