Algorithms: Homework 1

Yilin Zheng
jerryzyl18@gmail.com

Junfeng Liu
11510801@mail.sustc.edu.cn

July 25, 2017

Enter Fibonacci

Problem (a)

Prove $F_n \geq 2^{n/2}$ for all $n \geq 6$.

Solution

We can analysis some cases:

$$
\begin{aligned}
&F_0 = 1, \quad F_1 = 1 \\
&F_2 = F_0 + F_1 = 2 \\
&F_3 = F_1 + F_2 = 3 \\
&F_4 = F_2 + F_3 = 5 \\
&F_5 = F_3 + F_4 = 8 \\
&F_6 = F_4 + F_5 = 13 \\
&\ldots
\end{aligned}
$$

Proof.

By mathematical induction:

- **Base case**:
  For $n = 6$ and $n = 7$,
  $$F_6 = 13 \geq 2^{\frac{6}{2}} = 8, \quad F_7 = 21 \geq 2^{\frac{7}{2}} \approx 11.31$$
  which is **true**.
- **Induction hypothesis**:
  Suppose for $n = k - 1$ and $n = k$,
  $$F_k \geq 2^{\frac{k}{2}}, \quad F_{k-1} \geq 2^{\frac{k-1}{2}}$$
- **Induction step**:
  For $n = k + 1$,
  $$
  \begin{aligned}
  F_{k+1} &= F_{k-1} + F_k \\
  &\geq 2^{\frac{k-1}{2}} + 2^{\frac{k}{2}} \\
  &\geq 2^{\frac{k-1}{2}} + 2^{\frac{k-1}{2}} \\
  &= 2 \times 2^{\frac{k-1}{2}} \\
  &= 2^{\frac{k+1}{2}}
  \end{aligned}
  $$

So **hypothesis holds for all** $n \geq 6$. ■

Problem (b)

Show that $F_n \geq 2^n$ for all n.

Solution

Proof.

- **Base case**:
  For $n = 0$ and $n = 1$,
  $$F_0 = 1 \leq 2^0 = 1, \quad F_1 = 1 \leq 2^1$$
  which is **true**.
- **Induction hypothesis**:
  Suppose for $k > 0, n \in [0, k]$ holds
  $$F_k \leq 2^k, \quad F_{k-1} \leq 2^{k-1}$$
- **Induction step**:
  For $n > k + 2$,
  $$\begin{aligned} F_{k+1} &= F_k + F_{k-1} \\ &\leq 2^k + 2^{k-1} \\ &\leq 2^k + 2^k \\ &= 2^{k+1} \end{aligned}$$

So **the hypothesis holds for all** n. ■

Problem (c)

Find a if a and b are integers such that $x^2 - x - 1$ is factor of $ax^{17} + bx^{16} + 1$.

Solution

By analysis:
Let
$$F(x) = ax^{17} + bx^{16} + 1$$
Since $x^2 - x - 1$ is a **factor**, then there must have
$$f(x) = c_1x^{15} + c_2x^{14} + \ldots + c_{15}x - 1$$
which is
$$\begin{aligned} F(x) &= f(x)(x^2 - x - 1) \\ &= (c_1x^{15} + c_2x^{14} + \ldots + c_{15}x - 1)(x^2 - x - 1) \end{aligned}$$
Since there is **no** x **item in** $F(x)$, so, $c_{15} = 1$.
Then consider c_{14}, to **keep only 1 item of** x^2 **in** $F(x)$, c_{14} must be -2
For c_{13}, analyze in the same way, we can get $c_{13} = 3$
So there are what we find:
$$\begin{aligned} c_{15} &= 1 \\ c_{14} &= -2 \\ c_{13} &= 3 \\ c_{12} &= -5 \\ c_{11} &= 8 \\ &\ldots \end{aligned}$$
which is a little like **Fibonacci numbers**.
So, we can calculate and get the result $c_1 = 987$. ■

Testing i*Phone

Problem

Describe an efficient testing strategy and prove that it is indeed efficient.

Solution

The problem of increasing the height little by little or binary search based strategy is that **the tests for different cases differ greatly**.

So, the efficient strategy is to make the number of test in different cases more equal. Let generate the efficient strategy step by step.

Suppose the unit to increase or decrease height is $\delta(\delta > 0)$, we have n devices and the given maximum height H can be divided by δ, assume $H = m\delta(m > 0)$.

Suppose a base case with 2 devices:

After first test at $k\delta(0 < k < m)$, there are two cases might happen:

(i) Device breaks, then we leave only 1 devices and in worst case need to test from δ to $(k-1)\delta$ to find the max height, which takes $k-1$ tests, in this case, the total tests is $1 + k - 1 = k$ times.

(ii) Device didn't break, then we still have 2 devices and in worse case need to test from $(k+1)\delta$ to $m\delta$, which takes $(m-k)\delta$ tests.

But to minimize the number of tests, we should take the case which yields the mini tests. As analysis before, to minimize the number is actually make all cases take the same number of tests. So in this case:

(i)if device dosen't break, rather than add-on $k\delta$ or test from $(k+1)\delta$ to $m\delta$, we add $(k-1)\delta$ to start next test, so if the second device broke, we need to take $(2k-1) - k + 1 = k$ tests.

(ii)if not, then we add $k-2$ to continue testing...

Similarly, the number of tests is

$$k + (k-1) + (k-2) + ... + 2 + 1$$

let $\delta = 1$, then

$$H = k\delta + (k-1)\delta + (k-2)\delta + ... + 2\delta + \delta = \frac{(1+k)k}{2}$$

so

$$k = \frac{-1 + \sqrt{1+8H}}{2}$$

In this case, $n = 2$,

$$g_2(H) = \frac{-1 + \sqrt{1+8H}}{2} \approx H^{\frac{1}{2}}$$

Further, if we have 3 devices, then we can test in the same way, but to keep the total tests in different cases the same, we should modify add-on height in i times $k - i$ to be

$$k' = \frac{(1+k')k'}{2}$$

so for $n = 3$,

$$H = \sum_{k'=1} \frac{(1+k')k'}{2}$$

then

$$g_3(H) = k' \approx H^{\frac{1}{3}}$$

So, continue increasing devices to $4, 5, 6, ...$ and test in similar way, then we can find the relation about n and H and derive the function

$$g_n(H) = H^{\frac{1}{n}}$$

then,

$$lim_{H\to\infty} g_n(H)/g_{n-1}(H) = lim_{H\to\infty} \frac{H^{\frac{1}{n}}}{H^{\frac{1}{n-1}}} = lim_{H\to\infty} \frac{1}{H} = 0$$

Then we prove this **strategy is efficient**. ■

Reference: The Two Egg Problem

Studyitis

Problem

Prove that if less than n students are initially infected then the whole class will not be completely infected.

Solution

- **Base case**:

  For $n = 1$, which is 1×1 grid, if no one infected, the statement is true.

- **Induction hypothesis**:

  Suppose for $n = k(k > 0)$, k infected students can infect the whole class which is $k \times k$ grid.

- **Induction step**:

  For $k + 1$, since k infected students can infect $k \times k$ gird students, then the infection will stop and leaves $4k$ uninfected students, which are not adjacent to at least two infected students.

  If we make one more student infect, then the infection is able to continue since at least one uninfected student is adjacent to two infected students which cause himself or herself to be infected in next time step.

  So, need at least $k + 1$ students to infect the whole class which is $(k + 1) \times (k + 1)$ grid.

The hypothesis is proven. ■

How Complex is Complex

Problem (a)

i

Express the running time of your algorithm in $O(T(n))$ and judge whether it is polynomial in the size of the input.

Solution

According to **Fibonacci definition**: $F_n = F_{n-1} + F_{n-2}$, the algorithm is:

Algorithm 1 Fibonacci(n)

Input: An nonnegative integer n
Output: The nth Fibonacci number F_n

```
1: function FIBONACCI(n)
2:     if n == 0 or n == 1 then
3:         return 1
4:     else
5:         return FIBONACCI(n − 2)+FIBONACCI(n − 1)
6:     end if
7: end function
```

The algorithm calculates Fibonacci number recursively and takes running time $O(2^n)$, so the running time of this algorithm is **not polynomial in the size of the input**. ■

ii

Try to find an algorithm bounded by a polynomial in the size of the input.

Solution

The following algorithm computes F_n in $O(n)$ running time. ■

Algorithm 2 Fibonacci(n)

Input: An nonnegative integer n
Output: The nth Fibonacci number F_n

```
function FIBONACCI(n)
    if n == 0 or n == 1 then
        return 1
    else
        for i := 1 to n do
            tmp ← p
            p ← q
            q ← tmp + q
        end for
    end if
    return q
end function
```

Problem (b)

Compute the running time of three algorithms and choose one from them.

Solution

- For i, the recurrence is $T(n) = 5T(\frac{n}{2}) + O(n)$
- For ii, the recurrence is $T(n) = 2T(n-1) + O(n)$
- For iii, the recurrence is $T(n) = 9T(\frac{n}{3}) + O(n^2)$

Use **The Master Theorem**:

- For i, $a = 5$, $b = 2$, $f(n) \in O(n)$, then $K > 1$, so,

$$T(n) = \Theta(n^{log_2 5}) < \Theta(n^2)$$

.

- For ii, none of three cases apply, so,

$$\begin{aligned} T(n) &= 2T(n-1) + O(n) \\ T(n-1) &= 2T(n-2) + O(n) \\ T(n-2) &= 2T(n-3) + O(n) \\ &\cdots \\ T(2) &= 2T(1) \\ T(1) &= 1 \end{aligned}$$

so, we can derive that

$$T(n) = 2^n + O(n^2) \in \Theta(2^n)$$

.

- For iii, $a = 9$, $b = 3$, $f(n) \in O(n^2)$, coefficient

$$k = n^{log_b a} = n^2$$

then $af(n/b) = f(n)$, so,

$$T(n) = \Theta(n^2 log_3 n)$$

.

Compare three results, I will choose algorithm **(i)** since it has minimum running time. ■

Problem (c)

i

Judge the statement is true or not.

Proof.

Since $f(n) \in O(g(n))$, there exists a constant C, for $n \to +\infty$,

$$f(n) \leq Cg(n)$$

so, there exists another constant C', that

$$2^{f(n)} \leq C' \cdot 2^{Cg(n)} = C' \cdot (2^C)^{g(n)}$$

The inequation holds when $C = 1$ **but false otherwise**.

That is to say:

(i)When $f(n) \leq g(n)$,

$$2^{f(n)} \in O(2^{g(n)})$$

is **true**.

(ii)When $C > 1$, take $C = 2$ for a counterexample: let $f(n) = 2n$, $g(n) = n$, then:

$$2^{f(n)} = 2^{2n} \in \Theta(4^n)$$
$$O(2^{g(n)}) = O(2^n) \in \Theta(2^n)$$

when $n \geq 1$, $O(4^n) > O(2^n)$, which makes the statement false. ■

ii

Prove $O(f + g) = O(f) + O(g)$.

Proof.

Let $F \leq c_1 f, G \leq c_2 g$, $c_3 = max\{c_1, c_2\}$, then, $F \leq c_3 f, G \leq c_3 g$, so

$$\begin{aligned} O(f) + O(g) &= F + G \leq c_3 f + c_3 g \\ &= c_3(f + g) = O(f + g) \end{aligned}$$

The statement is proven. ■

Problem (d)

Show that $5n + 8n^2 + 100n^3 \in O(n^3 log(n))$. Also exam whether is $5n + 8n^2 + 100n^3 \in \Theta(n^3 log(n))$.

Proof. 1

Take the limit:

$$\begin{aligned} lim_{n\to\infty} \frac{5n + 8n^2 + 100n^3}{n^3 logn} &= lim_{n\to\infty} \frac{\frac{5}{n^2} + \frac{8}{n} + 100}{logn} \\ &= lim_{n\to\infty} \frac{100}{\infty} = 0 \end{aligned}$$

So, there exists a constant $C > 0$, for $n \to \infty$, $5n + 8n^2 + 100n^3 \leq Cn^3 logn$, which means $5n + 8n^2 + 100n^3 \in O(n^3 log(n))$. ■

Proof. 2

Take the limit reversedly:

$$lim_{n\to\infty}\frac{n^3logn}{5n+8n^2+100n^3} = lim_{n\to\infty}\frac{logn}{\frac{5}{n^2}+\frac{8}{n}+100}$$
$$= lim_{n\to\infty}\frac{\infty}{100} = \infty$$

There doesn't exist any constant C which can make $5n + 8n^2 + 100n^3 \geq Cn^3logn$ true, so, $5n + 8n^2 + 100n^3 \notin \Theta(n^3log(n))$. ■