

MATH 22

Lecture P: 10/23/2003

ANALYSIS OF ALGORITHMS; REVIEW

I ain't lookin' to block you up,
Shock or knock or lock you up,
Analyze you, categorize you,
Finalize you, or advertise you.

—Bob Dylan,
'All I Really Want To Do'

Administrivia

- <http://larry.denenberg.com/math22/LectureP.pdf>
- Exam Monday, 11:50 – 1:20, Robinson 253
- “All questions will be from the homework and projects and the two handouts (big O and induction)”
- Response to grader’s complaint: Turn proofs upside down!

Algorithm Analysis

The goal is to be able to find the complexity of a computer program (or algorithm, which for us is the same thing). Recall that *the complexity of a program Q* is a function f_Q such that

$f_Q(n)$ = the worst-case running time of Q over all inputs of size n

where time is measured not in seconds, but in some kind of “basic operations”.

In fact, we won't try to find f_Q itself; we just want the *rate of growth* of f_Q . It's enough to know that f_Q is in $O(n)$ or $O(n^2)$ or $O(\log n)$ or $O(2^n)$ or whatever.

Our programs are written in Grimaldi's pseudocode: Variables aren't declared, input is supplied magically in some variable and output is just left in another variable (or perhaps **returned**). We have loops of the forms

```
for <var> := <start> to <end> do  
    while <condition> do
```

with scope indicated by indentation. Assignment is performed by the `:=` operator.

Example: TRIANGLE

Here is an example. The following program solves the problem **TRIANGLE**: Given a positive integer N , compute $1 + 2 + 3 + \dots + N$. [Why “triangle”?]

```
Input: Positive integer N
    sum := 0
    for i := 1 to N do
        sum := sum + i
```

(Is it clear that this algorithm solves the problem?)

What is the complexity of this program? The first statement is executed once. The second and third statements are each executed N times. The total time is

$$f(N) = c_1 + Nc_2 + Nc_3 = c_1 + (c_2 + c_3)N$$

where c_j is the number of basic operations in step j . We have $f \in O(N)$ since we can ignore constant factors and all powers of N except the highest.

Of course we don't usually write all these details. We ignore the first statement and simply note that the program has a single loop that does constant work and is executed N times, so the time is obviously in $O(N)$.

A Worse Algorithm

Here's another algorithm that solves the same problem:

```
Input:  Positive integer N
        sum := 0
        for i := 1 to N do
            for j := 1 to i do
                sum := sum + 1
```

Is it just as clear that this algorithm solves the problem?

What is the time complexity? The “outer” loop (on **i**) executes N times. But the number of times that the “inner” loop (on **j**) executes changes each time, depending on the outer loop: It executes **i** times, but **i** varies from 1 to N .

So how many times is the final step executed? It's executed $1 + 2 + 3 + \dots + N$ times, which we know is $N(N+1)/2 = (1/2) N^2 + (1/2) N$. Ignoring constant factors and powers of N other than the largest, we find that the algorithm's time complexity is in $O(N^2)$.

In both examples we've seen, the number of times things execute hasn't depended on the input: worst, average, and best case are all the same.

A Better Algorithm

One last algorithm for the same problem:

Input: Positive integer N

sum := $N * (N+1) / 2$

This algorithm runs in *constant time*, that is time $O(1)$, time *independent of N* . It beats the other two handily.

You may see a loophole in what we've done so far: Why not just wildly overestimate? The time complexity of all three algorithms we've seen is in $O(2^n)$; why not just say that and be done, eh, grader?

Grimaldi's response: No, we want the 'best "big-Oh" form', that is, if the algorithm is $O(n)$ and we answer $O(n^2)$, we're wrong because that answer isn't 'best'.

Denenberg's response: No, because what we're really looking for is the *exact order* of the algorithms; we want \square , not O . We're not proving it, but in fact the three algorithms we've seen are in $\square(n)$, $\square(n^2)$, and $\square(1)$. The complexities are no bigger and also no smaller.

Searching

We now consider an algorithm for **SEARCH**: Given a sequence S of number $s_1, s_2, s_3, \dots, s_n$ and a target number s , determine whether s is found anywhere in S .

```
for i := 1 to n
    if s = si return "yes"
return "no"
```

How many times does the loop execute? It depends on whether s is in the sequence, and where it is! In the best case $s = s_1$ and the loop executes once. But we've defined complexity to measure the worst case, which happens either when $s = s_n$ or s isn't in the sequence at all, in which case the loop executes n times. So the complexity of the algorithm is in $O(n)$.

[As we said, we sometimes prefer to consider average case complexity. Here the average number of times through the loop is $n/2$ assuming that s is in the sequence and is equally likely to be located anywhere. But maybe s is rarely in the sequence, or is much more often one of the first members! We can get any answer from $O(n)$ to $O(1)$ depending on these assumptions.]

A Better Algorithm

Suppose that the members of sequence S are distinct and *ordered*, that is, $s_1 < s_2 < s_3 < \dots < s_n$. Then there is a much better algorithm known as *binary search*:

```
L := 1,   R := n           (searching from L to R)
while L ≤ R           (while anything remains...)
    M := ⌊(L+R)/2⌋       (M is the middle)
    if s = sM return "yes"           (found)
    if s < sM then R := M-1         (cut in half)
    if s > sM then L := M+1       (cut in half)
return "no"           (not there)
```

[Blackboard explanation of how this works]

How many times does the loop execute? The key point is that each time through the loop, the number of elements remaining is *cut in half*! So the answer is this: The loop executes *as many times as you have to cut n in half to get down to 1*. This number is $\log_2 n$ or $\lg n$.

[By definition, $2^{\lg n} = n$, which says that if you start with 1 and double $\lg n$ times you get n . Binary search does the same thing backwards, halving. Learn this!]

So the worst-case complexity of this program is $O(\lg n)$.

What You Need

I. SETS

- Elements, subsets, proper subsets, set equality
- Union, intersection, complement, [sym. diff.]
- Cardinality, power set, null set
- [Membership tables], Venn diagrams
- Ordered pairs, Cartesian product
- Elementary probability (count and divide)

II. MATHEMATICAL INDUCTION

III. RELATIONS and FUNCTIONS

- Relations and binary relations and their properties
- Domain, codomain, range, image, preimage
- Injective (1-1), surjective (onto), bijective (both)
- Floor and ceiling functions
- [Functions of multiple arguments, projections]
- Composition of functions
- Inverses of functions
- Growth rates of functions, O and Ω notation
- Elementary algorithm analysis

Requests & Examples

- Grimaldi section 5.6 number 18
- If $n \geq 14$, then n can be written as a sum of 3s and 8s. (Also, the problem on the Fundamental Theorem of Arithmetic from Project 4.)
- Suppose $f: A \rightarrow B$ and $g: B \rightarrow C$ are both onto. What are the domain and codomain and range of $f \circ g$? What are the domain and codomain and range of $g \circ f$? Pick one of these functions and show that it is onto.
- Prove that $\lceil x + y \rceil \geq \lceil x \rceil + \lceil y \rceil$ for all real x and y .
- Prove that $A \cap B = A$ if and only if $\overline{B} \subseteq \overline{A}$ (remember that Grimaldi uses overline for complement!)
- What is the probability that a two-digit number (10-99) contains a 7? What about a three-digit number?
- Suppose f is a function whose domain and codomain are the digits 0 through 9. What is the probability that the image of every even digit is also an even digit?

Requests & Examples

Grimaldi section 5.6 number 18

The key thing is to regard f , g , and h as functions whose input is a single argument, namely an ordered pair of sets, and whose output is a set.

All three functions are onto and none are one-to-one. Hence none are invertible, which requires one-to-oneness.

All the sets of part (d) that deal with f^{-1} and h^{-1} are infinite; there are lots of ways to make small sets with intersections and symmetric differences!

$g^{-1}(0)$ has a single element: the ordered pair $(0,0)$

$g^{-1}(\{2\})$ has two elements: the ordered pair $(0,\{2\})$ and the ordered pair $(\{2\},0)$ [these are different!]

$g^{-1}(\{8,12\})$ has four elements: $(0,\{8,12\})$, $(\{8,12\},0)$, $(\{8\},\{12\})$, and $(\{12\},\{8\})$

Requests & Examples

If $n \geq 14$, then n can be written as a sum of 3s and 8s.

Here is a *flawed* proof using the strong form of the Principle of Mathematical Induction:

Base case: If $n = 14$, then n can be written $3 + 3 + 8$.

Inductive case: Assuming that every number from 14 up through n can be written as a sum of 3s and 8s, we prove that $n+1$ can be so written. We can write

$$n+1 = 3 + (n-2)$$

Now $n-2$ can be written with 3s and 8s by the Inductive Hypothesis, so $n+1$ can be so written by just adding another 3. Done.

What's wrong with this proof? The Inductive Hypothesis applies only to numbers 14 or greater. We've applied it to $n-2$. So we must have $n-2 \geq 14$, which is to say $n \geq 16$. That is, our Inductive Case can only be used to prove the theorem for 17 and higher! So we must explicitly show that the Theorem is true for $n=15$ and $n=16$; we need two more base cases! These are easy ($15=3+3+3+3+3$ and $16 = 8+8$) so we're done.

Requests & Examples

Suppose $f : A \rightarrow B$ and $g : B \rightarrow C$ are both onto.

What are the domain and codomain and range of $f \circ g$?

What are the domain and codomain and range of $g \circ f$?

Pick one of these functions and show that it is onto.

First of all $f \circ g$ isn't defined at all; g takes something in B to something in C , but f can't then operation on something in C ! So forget $f \circ g$.

But $g \circ f$ is OK: f takes something in A to something in B , which g then takes to something in C . So the domain of $g \circ f$ is A and the codomain is C .

It turns out that $g \circ f$ must be onto, as we will show, so the range of $g \circ f$ is all of C .

To show $g \circ f$ is onto, we must show that for any $c \in C$ there is an $a \in A$ such that $(g \circ f)(a) = c$, which is to say that $g(f(a)) = c$. So given such a $c \in C$, the fact that g is onto means that there is some element of B , call it b_1 , such that $g(b_1) = c$. Now by the onto-ness of A there is an element of A , call it a_1 , such that $f(a_1) = b_1$. But if $f(a_1) = b_1$, and $g(b_1) = c$, then $g(f(a_1)) = c$, so we have found the necessary a and the proof is complete.

Requests & Examples

Prove that $\lfloor x + y \rfloor \geq \lfloor x \rfloor + \lfloor y \rfloor$ for all real x and y .

We can write $x = \text{floor}(x) + r_x$, where r_x is a number at least 0 and less than 1. Similarly, $y = \text{floor}(y) + r_y$ where $0 \leq r_y < 1$. (All we've done is to separate out the fractional parts of x and y .)

So

$$x+y = \text{floor}(x) + \text{floor}(y) + r_x + r_y$$

and thus

$$\text{floor}(x+y) = \text{floor}(\text{floor}(x) + \text{floor}(y) + r_x + r_y)$$

If we now throw away r_x and r_y the right-hand side may get smaller but can't get bigger, since r_x and r_y are nonnegative. So we have

$$\text{floor}(x+y) \geq \text{floor}(\text{floor}(x) + \text{floor}(y))$$

Finally, $\text{floor}(x)$ and $\text{floor}(y)$ are integers, so taking further floors of them (even after adding) does nothing.

That is,

$$\text{floor}(\text{floor}(x) + \text{floor}(y)) = \text{floor}(x) + \text{floor}(y)$$

and we're done.

Requests & Examples

Prove that $A \cap B = A$ if and only if $\overline{B} \subseteq \overline{A}$
(remember that Grimaldi uses overline for complement!)

This is an “if or only if”, so we must prove two things:

- If $A \cap B = A$, then $\overline{B} \subseteq \overline{A}$

So assume $A \cap B = A$. To prove $\overline{B} \subseteq \overline{A}$ we must prove that any $x \in \overline{B}$ is also an element of \overline{A} . So let x be any element in \overline{B} . By definition of \overline{B} we have $x \notin B$. But if $x \in B$, then $x \in A \cap B$ (since anything not in B can't be in the intersection of B with anything!). And if $x \in A \cap B$ then $x \in A$, since $A \cap B = A$ by assumption. Finally, if $x \in A$ then $x \notin \overline{A}$.

- If $\overline{B} \subseteq \overline{A}$, then $A \cap B = A$.

So assuming $\overline{B} \subseteq \overline{A}$ we must prove $A \cap B = A$. One way to prove two sets equal is to prove that each is a subset of the other, that is, $A \cap B \subseteq A$ and $A \subseteq A \cap B$. The first of these is always true; anything in $A \cap B$ is by definition in A ! So we only need to prove $A \subseteq A \cap B$, that is, any $x \in A$ is an element of $A \cap B$. It suffices to show that $x \in B$, that is, we must show that if $x \in A$ then $x \in B$. But we know that $\overline{B} \subseteq \overline{A}$, i.e., that if $x \in \overline{B}$ then $x \in \overline{A}$, and this is the contrapositive of (hence equivalent to) the thing we want to prove!

Requests & Examples

What is the probability that a two-digit number (10-99) contains a 7? What about a three-digit number?

There are 9 two-digit numbers that contain a seven in the unit's place (17, 27, . . . , 97). There are ten that contain a seven in the ten's place (70, 71, 72, . . . 79). But one of these numbers is double-counted, namely 77. So there are 16 numbers that contain a seven.

There are 90 two-digit numbers total. So the answer is 16/90.

This is a use of the counting rule that we proved with Venn Diagrams: $|A \cup B| = |A| + |B| - |A \cap B|$. Here A is the set of numbers with a seven in the unit's place and B is the set of numbers with a seven in the ten's place.

To do it for three-digit numbers, you must use the formula for the size of the union of three sets that we learned in Lecture J. Check it out. The answer is

$$(90 + 90 + 100 - 9 - 10 - 10 + 1) / 900$$

Requests & Examples

Suppose f is a function whose domain and codomain are the digits 0 through 9. What is the probability that the image of every even digit is also an even digit?

To build a function from digits to digits, we have to pick a value of the function for each digit. That is, we have to pick a value for $f(0)$ and there are ten choices, for $f(1)$ and there are ten choices, etc. So the total number of functions from digits to digits is 10^{10} . (Recall the result from the notes, or from Grimaldi, that the number of functions from finite set S to finite set T is $|T|^{|S|}$.)

How many such functions take even digits to even digits? There are now only five choices for $f(0)$; it must be 0, 2, 4, 6, or 8. Similarly, there are five choices for $f(2)$, $f(4)$, $f(6)$, and $f(8)$. But $f(1)$ can still be any of the ten digits, as can $f(3)$, $f(5)$, $f(7)$, and $f(9)$. So the answer is $(5^5)(10^5)$.

So the probability that a function from digits to digits takes even digits to even digits is $(5^5)(10^5)$ divided by 10^{10} . This is a perfectly acceptable answer and doesn't need to be simplified to $1/32$.