Complexity

T Weinhart

Created by : R van Damme (2009), updated by A Thornton (2011)

Last update: November 2013

Preview: today's lecture

- **Efficiency**
  What is an efficient algorithm?
- **Debugging**
  Why does this code not work?
- **Optimisation**
  What can I do to improve the speed of my code?

Efficiency/complexity of an algorithm

A non-formal definition of

'time is money, and memory is money, too'

Relevant questions:

1. How many steps does the algorithm need to solve the problem, as a function of $N \gg 1$?
   Time efficiency
2. And how many memory does your algorithm take, as a function of $N \gg 1$?
   Memory efficiency
3. Usually a worst case analysis.

Algorithm GCD

GCD(n, m) is largest $z \in \mathbb{N}$ st **div**(n, z) and **div**(m, z) are both true; the function **div**(n, z) return true if and only z divides n.

---

input: $n, m \in \mathbb{N}$
output: largest $z \geq 1$ st **div**(n, z) and **div**(m, z) **true**

```
for k = 1 to min(m, n) do
    if div(n, k) and div(m, k) then
        z ← k
    end if
end for
```

Is this a clever design? Time/memory is money. Not really....

Algorithm GCD v2

Idea: change the order of the loop

---

input: $n, m \in \mathbb{N}$
output: largest $z \geq 1$ st **div**(n, z) and **div**(m, z) **true**
  for $k = \min(m, n)$ downto 1 **do**
    if div(n, k) and **div**(m, k) **then**
      return $z \leftarrow k$
    end if
  end for

Due to the for-loop you have to do something $O(N)$ times. For memory you need not worry here: you need $O(1)$ memory.

But is $O(N)$ good or bad?

Efficiency/complexity of an algorithm

Suppose we have a problem in which have a parameter N. Examples are

- Compute the N-th prime number
- Compute the inverse of a $N \times N$ matrix
- Compute the GCD of number a, b with $N = \min(a, b)$.
- Find all permutations of an array A of length N.
- Sort the elements of an array A of length N.

Complexity of an algorithm: calculus

We are mainy interested in the order of complexity, i.e., what happens for large N.

- $$c = 14N^2 + 1000N + 1000000$$
  $$\Rightarrow c \approx 14N^2 \Rightarrow c = O(N^2)$$
- $$c = 14N^2 + 0.0001 \cdot 2^N$$
  $$\Rightarrow c \approx 0.0001 \cdot 2^N \Rightarrow c = O(2^N)$$

but everything is relative: $O(N^3) > O(N^2)$ but

$$N^3 < 10^{10} N^2 \text{ if } N < 10^{10}$$

Complexity of an algorithm: an impression

For which N can we solve a problem taking $f(N)\mu$seconds?

$f(N)\downarrow, T\rightarrow$	1 second	1 month	1 century
$\ln N$	10^{434294}	$10^{112569129709}$	$10^{1369591078130094}$
N	10^{6}	10^{12}	10^{15}
$N \ln N$	10^{5}	10^{11}	10^{14}
N^2	10^{3}	10^{6}	10^{7}
2^N	19	41	51
$N!$	9	14	17

Ex: permutations $\mathcal{O}(N!N)$, sorting $\mathcal{O}(N \log N)$, GCD $\mathcal{O}(\log N)$

Algorithm GCD v3: Euklid's algorithm

input: $n, m \in \mathbb{N}$
output: largest $z \geq 1$ s.t. $\text{mod}(n, z) = 0$ and $\text{mod}(m, z) = 0$

```
while n ≠ 0 do
    k ← mod(m, n)
    m ← n
    n ← k
end while
return z ← n
```

The product $m \cdot n$ will more than half with each iteration.
$\Rightarrow$ the algorithm requires only $O(\log N)$ operations.

Complexity of an algorithm: calculus

for $k = 1$ to N **do**
  $z \leftarrow k$, $u \leftarrow k \cdot k$
end for

Every not nested for-loop $O(N)$, "no matter what" happens in between: $2N = O(N)$.

Complexity of an algorithm: calculus

---

for $k = 1$ to N **do**
  $z \leftarrow k$, $u \leftarrow k \cdot k$
end for
for $m = 0$ to $N + 3$ **do**
  $z \leftarrow z + m$, $u \leftarrow u \cdot m$
end for

A sequence of not nested for-loop $O(N)$, "no matter what" happens in between

Complexity of an algorithm: calculus

```
for k = 1 to N do
    for m = 2 to 2 · N do
        z ← k, u ← k · m
    end for
end for
```

Every nested double for-loop $O(N^2)$
$(N \cdot (2N - 1) = 2N^2 - N = O(N^2))$

Complexity of an algorithm: an example

```
for k = 1 to N do
    a ← 0
    for m = 1 to k do
        a ← a + x_m
    end for
    A_k = a/k
end for
```

What is computed here and what is the complexity of the algorithm?

Complexity of an algorithm: an example

```
for k = 1 to N do
    a ← 0
    for m = 1 to k do
        a ← a + x_m
    end for
    A_k = a/k
end for
```

A running average:

$$A_k = \frac{1}{k}\sum_{m=1}^{k} x_m \rightarrow A_1 = x_1, \;\; A_2 = \frac{1}{2}(x_1 + x_2),$$

Efficiency of an algorithm: an example

```
for k = 1 to N do
    a ← 0
    for m = 1 to k do
        a ← a + x_m
    end for
    A_k = a/k
end for
```

What is the complexity of the algorithm?

$$c = \sum_{k=1}^{N} k = \frac{1}{2} N(N-1) = O(N^2)$$

Is this a clever algorithm?

Efficiency of an algorithm: an example

$s \leftarrow 0$
for $k = 1$ to N **do**
  $s \leftarrow s + x_k$
  $A_k = s/k$
end for

The complexity of the algorithm is now $O(N)$.

Do it yourself

Describe an algorithm that finds the maximum of n numbers.
What is its complexity?
Does it cost twice as much to find the max and the min?

$z = a_1$
for $k = 2$ to n **do**
  if $a_k > z$ **then**
    $z = a_k$
  end if
end for

---

is an $O(n)$ algorithm: $n - 1$ comparisons are being made.

Do it yourself

So $n-1$ comparisons made for finding the max; if we would try to find the max and the min of this set, does it cost $2(n-1)$ comparisons?

Take n even (makes little difference).

Do it yourself

1. Take the first two numbers: a_1, a_2. You need one comparison to find out which is min and which is max. Suppose $a_1 > a_2$, then $m_x = a_1$, $m_n = a_2$.
2. Now take a_3, a_4; we need one comparison which is min and which is max of the two. Assume $a_4 > a_3$. Then

$$if(a_4 > m_x) \rightarrow m_x = a_4$$

$$if(a_3 < m_n) \rightarrow m_n = a_3.$$

Hence we need 3 comparisons to deal with 2 numbers. So in total we need $n/2 \cdot 3 = \frac{3}{2}n < 2n$ comparisons.

Computing a determinant

$$\begin{vmatrix} A_{1,1} & \cdots & A_{1,n} \\ & \vdots & \\ A_{n,1} & \cdots & A_{n,n} \end{vmatrix} = A_{1,1} \cdot \begin{vmatrix} A_{2,2} & \cdots & A_{2,n} \\ & \vdots & \\ A_{n,2} & \cdots & A_{n,n} \end{vmatrix} -$$

$$\ldots \pm A_{1,n} \cdot \begin{vmatrix} A_{2,1} & \cdots & A_{2,n-1} \\ & \vdots & \\ A_{n-1,1} & \cdots & A_{n-1,n-1} \end{vmatrix}$$

One $n \times n$ determinant costs computing n $n-1 \times n-1$ determinants plus n multiplications plus $n-1$ additions

One $n \times n$ determinant costs computing n $n-1 \times n-1$ determinants plus n multiplications plus $n-1$ additions

We neglect multiplications and additions:

$$C(n) = nC(n-1), \;\; C(2) = 3$$

One by one:

$$C(2) = 3 \;\rightarrow\; C(3) = 9 \;\rightarrow\; C(4) = 36 \;\rightarrow\; C(5) = 180$$

$$C(n) = \frac{3}{2} n!$$

Thanks to Gauss, we can compute determinants in n^3 steps, however, so this is what Linear Algebra taught you!!

Debugging

- Check error messages and M-lint warnings
- Use the MATLAB debugger
- Example: surface normal, bubble sort

Optimisation

- Preallocate arrays
- Vectorize your code
- Use logical operators
- Use the MATLAB profiler
- Don’t change data types of a variable