INTRODUCTION

Gustavo Carvalho
(ghpc@cin.ufpe.br)

Universidade Federal de Pernambuco
Centro de Informática, 50740-560, Brazil

Agenda

Centro de Informática UFPE

Universidade Federal de Pernambuco

Mini bio

- Bachelor's degree: CC (UFPE: 2006)
- Master's degree: CC (UFPE: 2010)
- Doctor's degree CC (UFPE: 2016)
  - Bremen (DE) e York (UK)
- UFPE: lecturer (2017 – now)
- Research interests
  - **Formal methods**
    (theory and practice)
- Site: `http://cin.ufpe.br/~ghpc`

Centro de Informática UFPE

Universidade Federal de Pernambuco

IF672: overview

- Mentimeter: two questions
- Website: https://sites.google.com/a/cin.ufpe.br/if672/
- Classroom + Slack
- Syllabus
- Teaching dynamics
- Studying dynamics
- Assessment dynamics
- Teaching assistants

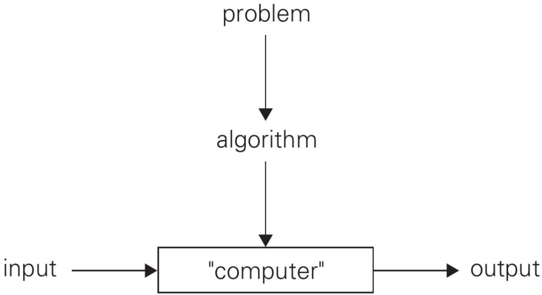
Agenda

1 IF672: algorithms and data structures

2 Algorithms: what? why?

What?[1]

"*An algorithm is a sequence of unambiguous instructions for solving a problem, i.e., for obtaining a required output for any legitimate input in a finite amount of time.*"



[1] A. Levitin. Introduction to the Design and Analysis of Algorithms. 2011.

Why?[2]

Practical standpoint
Theoretical standpoint
Developing analytical skills

"*Actually, a person does not really understand something until after teaching it to a computer, i.e., expressing it as an algorithm ...*"

Donald Knuth: https://en.wikipedia.org/wiki/Donald_Knuth

[2] Knuth, D.E. Selected Papers on Computer Science. CSLI Publications and Cambridge University Press, 1996.

Important points

1. The nonambiguity requirement cannot be compromised.
2. The range of valid inputs has to be specified carefully.
3. The same algorithm can be represented in several different ways.
4. There may exist several algorithms for solving the same problem.
5. Algorithms can solve the same problem with very different speeds.

Important points

1. The nonambiguity requirement cannot be compromised.
2. The range of valid inputs has to be specified carefully.
3. The same algorithm can be represented in several different ways.
4. There may exist several algorithms for solving the same problem.
5. Algorithms can solve the same problem with very different speeds.

GCD of two nonnegative and not-both-zero integers

- gcd(m, n) = the largest integer that divides both m and n evenly (with a reminder of zero)

gcd – 1st option (Euclid's algorithm)

In natural language

1. If $n = 0$, return the value of m as the answer and stop; otherwise, proceed to Step 2.
2. Divide m by n and assign the value of the remainder to r.
3. Assign the value of n to m and the value of r to n. Go to Step 1.

Centro de Informática UFPE

Universidade Federal de Pernambuco

gcd – 1st option (Euclid's algorithm)

In natural language

1. If $n = 0$, return the value of m as the answer and stop; otherwise, proceed to Step 2.
2. Divide m by n and assign the value of the remainder to r.
3. Assign the value of n to m and the value of r to n. Go to Step 1.

Algorithm: gcd(m,n)

---

input : two nonnegative, not-both-zero integers m and n

output : greatest common divisor of m and n

while $n \neq 0$ **do**
- $r \leftarrow m \bmod n$;
- $m \leftarrow n$;
- $n \leftarrow r$;

return m;

---

gcd – 1st option (Euclid's algorithm)

Computing: $gcd(60, 24)$

$m = 60$
$n = 24$
$r = 0$

Yes!

Algorithm: gcd(m,n)

input : two nonnegative, not-both-zero integers m and n
output : greatest common divisor of m and n

```
while n ≠ 0 do
    r ← m mod n;
    m ← n;
    n ← r;
return m;
```

gcd – 1st option (Euclid's algorithm)

Computing: $gcd(60, 24)$

$m = 60$
$n = 24$
$r = 12$

Remember that:
$q = \lfloor m/n \rfloor$
$m \bmod n = m - n * q$

$$\begin{aligned} q &= \lfloor 60/24 \rfloor \\ &= \lfloor 2.5 \rfloor = 2 \\ m \bmod n &= 60 - 24 * 2 \\ &= 12 \end{aligned}$$

Algorithm: gcd(m,n)

input : two nonnegative, not-both-zero integers m and n
output : greatest common divisor of m and n

```
while n ≠ 0 do
    r ← m mod n;
    m ← n;
    n ← r;
return m;
```

gcd – 1st option (Euclid's algorithm)

Computing: $gcd(60, 24)$

$m = 24$
$n = 12$
$r = 12$

Algorithm: gcd(m,n)

input : two nonnegative, not-both-zero integers m and n
output : greatest common divisor of m and n

1 **while** $n \neq 0$ **do**
2 $\quad r \leftarrow m \bmod n$;
3 $\quad m \leftarrow n$;
4 $\quad n \leftarrow r$;
5 **return** m;

gcd – 1st option (Euclid's algorithm)

Computing: $gcd(60, 24)$

$m = 24$
$n = 12$
$r = 12$

Yes!

Algorithm: gcd(m,n)

input : two nonnegative, not-both-zero integers m and n
output : greatest common divisor of m and n

while $n \neq 0$ **do**
  $r \leftarrow m \bmod n$;
  $m \leftarrow n$;
  $n \leftarrow r$;
return m;

Centro de Informática UFPE
Universidade Federal de Pernambuco

gcd – 1st option (Euclid's algorithm)

Computing: $gcd(60, 24)$

$m = 24$
$n = 12$
$r = 0$

Remember that:
$q = \lfloor m/n \rfloor$
$m \bmod n = m - n * q$

$$q = \lfloor 24/12 \rfloor$$
$$= \lfloor 2 \rfloor = 2$$
$$m \bmod n = 24 - 12 * 2$$
$$= 0$$

Algorithm: gcd(m,n)

input : two nonnegative, not-both-zero integers m and n
output : greatest common divisor of m and n

```
while n ≠ 0 do
    r ← m mod n;
    m ← n;
    n ← r;
return m;
```

Centro de Informática UFPE
Universidade Federal de Pernambuco

gcd – 1st option (Euclid's algorithm)

Computing: $gcd(60, 24)$

$m = 12$
$n = 0$
$r = 0$

Algorithm: gcd(m,n)

input : two nonnegative, not-both-zero integers m and n
output : greatest common divisor of m and n

```
while n ≠ 0 do
    r ← m mod n;
    m ← n;
    n ← r;
return m;
```

Centro de Informática UFPE

Universidade Federal de Pernambuco

gcd – 1st option (Euclid's algorithm)

Computing: $gcd(60, 24)$

$m = 12$
$n = 0$
$r = 0$

No!

Algorithm: gcd(m,n)

input : two nonnegative, not-both-zero integers m and n
output : greatest common divisor of m and n

```
while n ≠ 0 do
    r ← m mod n;
    m ← n;
    n ← r;
return m;
```

Centro de Informática UFPE
Universidade Federal de Pernambuco

gcd – 1st option (Euclid's algorithm)

Computing: $gcd(60, 24)$

$m = 12$
$n = 0$
$r = 0$

$gcd(60, 24) = 12$

Algorithm: gcd(m,n)

input : two nonnegative, not-both-zero integers m and n
output : greatest common divisor of m and n

```
while n ≠ 0 do
    r ← m mod n;
    m ← n;
    n ← r;
return m;
```

gcd – 2nd option (middle-school procedure)

1. Find the prime factors of m.
2. Find the prime factors of n.
3. Identify all common factors in the two prime expansions found in Steps 1 and 2. If p is a common factor occurring p_m and p_n times in m and n, respectively, it should be repeated $min(p_m, p_n)$ times.
4. Compute the product of all the common factors and return it as the greatest common divisor of the numbers given.

gcd – 2nd option (middle-school procedure)

1. Find the prime factors of m.
2. Find the prime factors of n.
3. Identify all common factors in the two prime expansions found in Steps 1 and 2. If p is a common factor occurring p_m and p_n times in m and n, respectively, it should be repeated $min(p_m, p_n)$ times.
4. Compute the product of all the common factors and return it as the greatest common divisor of the numbers given.

$$
\begin{aligned}
60 &= 2 * 2 * 3 * 5 \\
24 &= 2 * 2 * 2 * 3 \\
gcd(60, 24) &= 2 * 2 * 3 = 12
\end{aligned}
$$

Centro de Informática UFPE

Universidade Federal de Pernambuco

gcd – 2nd option (middle-school procedure)

What happens if m or n is equal to 0?

What happens if m or n is equal to 1?

How do we find the common elements of two lists?

How do we find the prime factors of an arbitraty number?

gcd – 2nd option | Sieve of Eratosthenes

Algorithm: sieve(n)

input : integer $n > 1$
output : array L with all prime numbers $\leq n$

```
for p ← 2 to n do A[p] ← p;
for p ← 2 to ⌊√n⌋ do
    if A[p] ≠ 0 then
        j ← p * p;
        while j ≤ n do
            A[j] ← 0;
            j ← j + p;
i ← 0;
for p ← 2 to n do
    if A[p] ≠ 0 then
        L[i] ← A[p];
        i ← i + 1;
```

Centro de Informática UFPE

Universidade Federal de Pernambuco

Remember: important points

1. The nonambiguity requirement cannot be compromised.
2. The range of valid inputs has to be specified carefully.
3. The same algorithm can be represented in several different ways.
4. There may exist several algorithms for solving the same problem.
5. Algorithms can solve the same problem with very different speeds.

Remember: important points

1. The nonambiguity requirement cannot be compromised.
2. The range of valid inputs has to be specified carefully.
3. The same algorithm can be represented in several different ways.
4. There may exist several algorithms for solving the same problem.
5. Algorithms can solve the same problem with very different speeds.

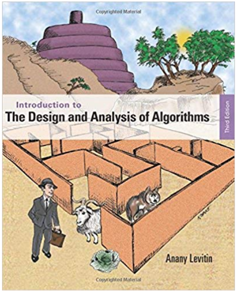
One more: proving an algorithm's correctness

```
Theorem findImpCorrectness :
  ∀ (n : nat) (l : list nat), findImp n l = true ↔ findSpec n l.
Proof. ... Qed.
```

Centro de Informática UFPE

Universidade Federal de Pernambuco

Bibliography



Chapter 1 (pp. 1–25). Anany Levitin.
Introduction to the Design and Analysis of Algorithms.
3rd edition. Pearson. 2011.

INTRODUCTION

Gustavo Carvalho
(ghpc@cin.ufpe.br)

Universidade Federal de Pernambuco
Centro de Informática, 50740-560, Brazil