LEARN CODING

ale66

PAC: THE PROBLEM-ALGORITHM-CODE TRIAD

MOTIVATIONS

- coding is often *problem solving + computer knowledge*
- we have worked on simple coding tasks: never in doubt about what was the desired result
- coding was essentially getting the computer to execute the right sequence of steps.
- but what if the solution is itself complex?

THE SEPARATION OF CONCERNS

Take problem solving with computers and break it down into three layers

Work out the layers separately; different teams might work on each layer

Replication is possible

COMPUTATIONAL PROBLEM

Identify the problem as an abstract input/output activity.

This the *what* of problem solving

EXAMPLE, I

Problem: **MAX**

Instance: a sequence of n integer numbers $a_1 \dots a_n$

Solution: element a_M such that $a_M \geq a_i$ for each element a_i of the sequence.

Algorithm: **enumeration**

```
input: sequence A
set a new variable M = 0

for each element a of A:
    if a > M then:
        set M = a

output: M
```

OBSERVATIONS

- the formulation is abstract from any real programming language (Python being the closests)
- the experience of what coding is help us understand the formulation
- we look at the algorithm to understand computational cost and thus *scalability*

ALGORITHM

A finite sequence of mathematically-rigorous instructions used to solve a class of specific problems or to perform a computation

- mathematical rigour is for effective implementability on computers
- the sequence shall be executed *deterministically*: at each step we know exactly what the next step will be.

IMPLEMENTATION

Re-formulation of the algorithm in a specific programming language/version.

```python
def max1(data):
  '''Classical implementation of max for a sequence of integers'''

  M = 0

  for element in data:
    if element > M
      M = element

  return M
```

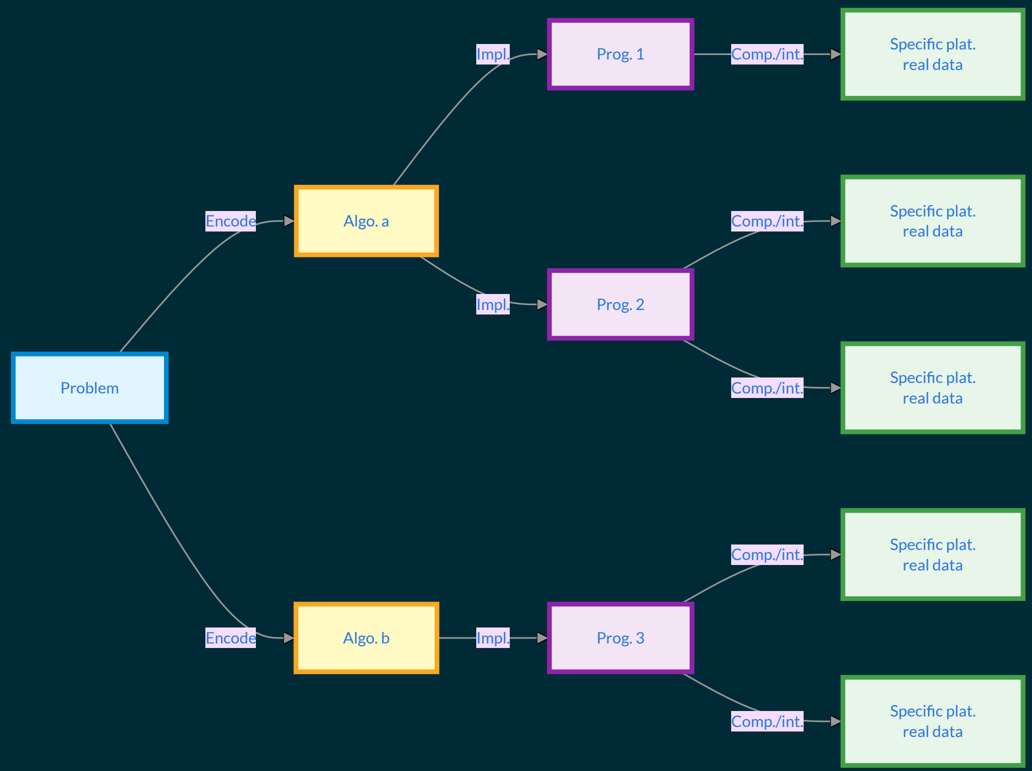
incorrect on all-negative inputs: `max1([-1, -6, -3, -11])` returns 0.

ANOTHER IMPLEMENTATION

```python
def mymax(data):
  '''Robust implementation of classical max '''

  M = data[0]

  for element in data[1:]:
    if element > M
      M = element

  return M
```

THE TRIAD

- for a fixed problem, several algorithms may exist
- for a fixed algorithm, several implementations, even in the same language, may exist



ALGORITHMICS

- discover new algorithms
- improve or re-pourpose existing ones
- provide bounds to the computational cost

SCALABILITY

The suitability of an algorithm for deployment to cases with an increasing volume of data.

- Can our algorithm run over the `unimi.it` web pages within a minute or two?
- Can it scale up to the whole `.it` domain, e.g., in an hour or two?
- Can it scale up to the whole *web*, e.g., in a day or two?

COMPUTATIONAL COSTS

Fact: scalability depends more on the intrinsic *complexity* of the problem than on the algorithm or the implementation.

Better implementation can contribute a scaling factor, e.g., 0.8, but cannot compress problem hardness

EXAMPLE: KNAPSACK

The Fourty thieves' vault contains bars of precious metal

Each bar is described by weight and value

Bars can be cut to weight

Ali Baba breaks in, determined to steal as much value as he can

However, he can only carry away a fixed quantity of weight on his shoulders.

Help Ali Baba to carry away the most value

KNAPSACK

FORMULATION, I

Instance:

A description of weight and value of each bar

the maximum weight Ali Baba can take on his shoulders

Solution:

An allocation: for each bar says what quantity is taken, so as to maximise the overall stolen value

Constraint

The sum of all taken wieights must not exceed what Ali Baba can carry

FORMULATION, II

Instance:

a sequence of *n* `weight/value` pairs (all non-negative int.):

an integer C

Solution:

An allocation: for each bar says what quantity is taken

Constraint

The sum of all taken quantities must not exceed C

AN EXAMPLE WITH FRUITS

FORMULATION, III

Instance:

a sequence of n `weight/value` pairs: $w_1, \ldots w_m$ and $v_1, \ldots v_n$

an integer C

Solution:

An allocation: for each bar says what quantity is taken: $\sigma : i \rightarrow [0..w_i]$

Constraint

The sum of all taken quantities must not exceed C: $\sum_{i=1}^{n} \sigma(i) \leq C$

FURTHER...

Instance:

- $w_1, \ldots w_n$ and $v_1, \ldots v_n$
- C

Solution:

- $\sigma : i \rightarrow [0..w_i]$
- $\sigma = MAXARG_\sigma \{\sum_{i=1}^{n} \nu(i)\}; \nu(i)$ is the val. of $\sigma(i)$

Constraints:

$\sum_{i=1}^{n} \sigma(i) \leq C$

ALGORITHM

Think of a step-by-step process that, for any input combination, will take the most value out of the vault

Write it down in English with some Maths

Test it on some examples, e.g.

Metal	Weight avail.	Total Val.
Palladium	1	72
Gold	3	162
Silver	12	48