Scuola universitaria professionale
della Svizzera italiana
SUPSI

Università
della
Svizzera
italiana

Istituto Dalle Molle di studi sull'intelligenza artificiale

Algorithms and Data Structures

Analysis of Algorithms

Matteo Salani
matteo.salani@idsia.ch

History

Muḥammad ibn Mūsā al-Khwārizmī

- Persian mathematician and astronomer
- In 825 A.D. writes the treaty Kitāb al-djabr wa 'l muqābala (The Compendious Book on Calculation by Completion and Balancing)
- It describes the Hindu numeric system Hindu based on positional notation (0 . . . 9)
- Translated in latin during XII century as "Algorithmo de Numero Indorum"
- "Algorithmo" was the latin translation of the author's name but the term has been misunderstood as the plural latin of "Algorismus"

Algorithms

An **algorithm** is a well defined computational procedure that processes a set of values (**input**) e produces a set of values (**output**).

It is therefore a sequence of computational steps that transforms input into output.

Problem solving

Anyway, it is more interesting to consider an algorithm as a tool to solve a well defined computational problem.
The computational problem defines the desired relationship among input and output

Example of computational problem

Given two sequences of integer numbers **A** and **B** of the same length **n**, determine the existence of a pair of numbers $(a \in A, b \in B)$ whose sum equals a given integer number **k**.

Example

$$\mathbf{A} = [6, 18, 22, 5, -10, 22]$$
$$\mathbf{B} = [2, -3, 12, 11, -1, 0]$$
$$\mathbf{n} = 6$$
$$\mathbf{k} = 33$$

The given example constitutes an **instance** of the problem.

An algorithm is said **correct** if, for any possible instance of the problem, the output satisfies the specification of the problem.

Pseudocode

Brute force algorithm:

Algorithm 1 Pair sum

```
function PairSum(A, B, n, k)
    for i ∈ {1 ... n} do
        for j ∈ {1 ... n} do
            if A[i] + B[j] = k then
                return True
            end if
        end for
    end for
    return False
end function
```

Is this algorithm efficient?

Exercises

- Given two sequences of integer numbers **A** and **B** of the same length **n**, determine all pair of numbers $(a \in A, b \in B)$ whose sum equals a given integer number **k**.
- Given two sequences of integer numbers **A** and **B** of the same length **n**, determine all pair of numbers $(a \in A, b \in B)$ whose sum is closer to a given integer number **k** (use absolute value $|a + b - k|$).
- Given two strings (sequences of characters) **A** e **B** of the same length **n**, determine whether string A is an anagram of string B

Think and report on the efficiency of proposed solutions.

Analysis of algorithms

The **analysis of algorithms** is the estimation/prediction of the resources that an algorithm requires for its execution for a given instance of a computational problem.
Resources can be:

- Memory requirements
- **Running Time**
- Communication bandwidth (for some type of algorithms)

In order to analyse an algorithm it is necessary to specify the reference architecture of the computing unit. The model commonly adopted is a **random-access machine (RAM)**

For personal interest you may find useful to explore the difference between CISC/RISC architectures

Analysis of algorithms

Given a reference model we refer to the so called "atomic" instructions:

- sum, subtraction, multiplication, division, rounding
- memory load/write/copy
- conditions, function call and return to routine

It is assumed that all atomic instructions are executed in constant time*.
It is assumed that an integer number of dimension n is represented by $c \log n$ bits with $c \geq 1$ constant.

*Further readings: floating point (memory/precision)? hierarchical memory (Cache, RAM)?

Analysis of algorithms

The Running Time (RT) depends on the input. It is therefore common to describe the RT as a **function of the input size**

How to define the **size of the input** depends on the problem. For many computational problems it is natural to define the input as a sequence of elements and to indicate with n its size.

The **running time** is the number of executed atomic instructions.

We assume that each line of the pseudocode is computed in a constant time c_i and we have to estimate how many times each line is executed.

Exercises

- Analyse the pseudocode of Pair Sum (Algo 1)
- Analyse the pseudocode of Insertion sort (cfr. cap. 2.2)
- Analyse the pseudocode of "Anagram" (cfr. slide 6)

Worst case and average case

For several reasons it make sense to study the RT in the worst case

- Guaranteed upper bound for any instance
- The worst case occurs often (always for some algorithms)
- Average case is sometimes as "worse" as the worst case

Classes of functions

$$1 < \log n < \sqrt{n} < n < n \log n < n^2 < n^3 < \cdots < 2^n < n^n$$

This holds for $n > n_0$

For example:

$\log n$	n	n^2	2^n
0	1	1	1
1	2	4	4
2	4	16	16
3	8	64	256

Valid also for $n^{100} < 2^n$ (holds for $n > n_0 = 997$)

Order of growth

In analysis of algorithm, rather than exact computational time, we are interested in establishing the order of magnitude/order of growth of running time for large input sizes.

It is therefore beneficial to use the **asymptotic notation**

Definizione

- $O()$, called **big-oh**: Upper bound of the function
- $\Omega()$, called **big-omega**: Lower bound of the function
- $\Theta()$, called **theta**: Average bound of the function

$O()$ - Big-oh

Definizione

The running time $T(n)$ is $O(f(n))$ iif there exist $n_0 \geq 0$ and $c > 0$ such that $T(n) \leq c \cdot f(n)$ for all $n \geq n_0$.

$\Omega()$ - Big-Omega

Definizione

The running time $T(n)$ is $\Omega(f(n))$ iif there exist $n_0 \geq 0$ and $c > 0$ such that $T(n) \geq c \cdot f(n)$ for all $n \geq n_0$.

$\Theta()$ - Theta

Definizione

The running time $T(n)$ is $\Theta(f(n))$ iif there exist $n_0 \geq 0$ and $c_1, c_2 > 0$ such that $c_1 f(n) \leq T(n) \leq c_2 \cdot f(n)$ for all $n \geq n_0$.

Example

Example

Given $T(0) = 0$ e $T(n) = (n+1) \cdot (n+2), n > 0$ show $T(n) = O(n^2)$

Example

Given $T(0) = 0$ e $T(n) = 2n + 3, n > 0$ show $T(n) = \Theta(n)$

Properties

In $O()$, $\Omega()$, $\Theta()$ notations constants do not matter.

Constants

If $T(n) = O(f(n))$ then $k \cdot T(n)$ is $O(f(n))$

Example:
$T(n) = 2 \cdot n^3 + 3$ is $O(n^3)$
$5 \cdot T(n) = 5 \cdot (2 \cdot n^3 + 3) = 10 \cdot n^3 + 15$ is $O(n^3)$

Properties

In $O()$, $\Omega()$, $\Theta()$ the transitive property holds.

Transitive prop.

If $T(n)$ is $O(f(n))$ and $f(n)$ is $O(g(n))$ then $T(n)$ is $O(g(n))$

Properties

In $\Theta()$ (**ONLY!**) the symmetric property holds.

Symmetric prop.

If $f(n)$ is $\Theta(f(n))$ then $g(n)$ is $\Theta(f(n))$

Properties

But it holds for $O() and \Omega()$ the transpose symmetric property.

Symmetric prop.

If $f(n)$ is $O(g(n))$ then $g(n)$ is $\Omega(f(n))$

Big-O notation

Simple instructions have constant time $O(1)$

For Loops

for (i=0; i $<$ n; i++)
    F

- If F is a simple instruction $\rightarrow O(n)$
- If F has RT $O(f(n)) \rightarrow O(n \cdot f(n))$

While and Do While

- We do not know the number of iterations
- The worst case is therefore the same of **for loops**

Big-O notation

If Then Else

If (C) Then B Else B'

- C is normally $O(1)$
- If B and B' are simple instructions $\rightarrow O(1)$
- If B has RT $O(f(n))$ and B' has RT $O(g(n)) \rightarrow O(max(f(n), g(n))$

```
if A[0] = 0 then
    for i ∈ {1 ... n} do
        A[i] = 0
    end for
else
    for i ∈ {1 ... n} do
        for j ∈ {1 ... n} do
            A[i] = A[i] + A[j]
        end for
    end for
end if
```

Big-O notation

Sequences

We use the rule of sums.

- $I_1, I_2, \cdots, I_m$
- $O(f_1), O(f_2), \cdots, O(f_m)$
- $O(f_1) + O(f_2) + \cdots + O(f_m)$

Function calls

We consider the RT of the called function.

Big-O notation

Recursion

We determine $T(n)$ with induction

- Let t the RT of a function call not using recursion t $(=O(1))$
- We express $T(n)$ as a function of recursive call $T(n')$ and develop the induction

```
function FACT(n)
    if n ≤ 1 then
        Return 1
    else
        Return n · Fact(n − 1)
    end if
end function
```

$$
\begin{aligned}
T(1) &= t \\
T(n) &= c + T(n-1) \\
&= 2c + T(n-2) \\
&\cdots \\
&= i \cdot c + T(n-i) \\
&= (n-1) \cdot c + T(1) \\
&= (n-1) \cdot c + t = O(n)
\end{aligned}
$$