

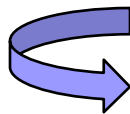


Introduzione

L'informatica e il concetto di
algoritmo – I linguaggi per la
programmazione di algoritmi

Che cos'è l'informatica?

- Pur avendo radici antichissime, l'informatica si è sviluppata nella seconda metà del ventesimo secolo con la diffusione dei calcolatori elettronici.



Il termine *Informatica*, coniato nel 1962, deriva dalla fusione delle parole *Informazione* e *Automatica*

Che cos'è l'informatica?

- Sono state date tante definizioni di questa disciplina:

- *scienza (e tecnologia)
dei calcolatori elettronici,
ma non solo*

- Focus sull'informazione:

- *scienza della rappresentazione e
dell'elaborazione dell'informazione*

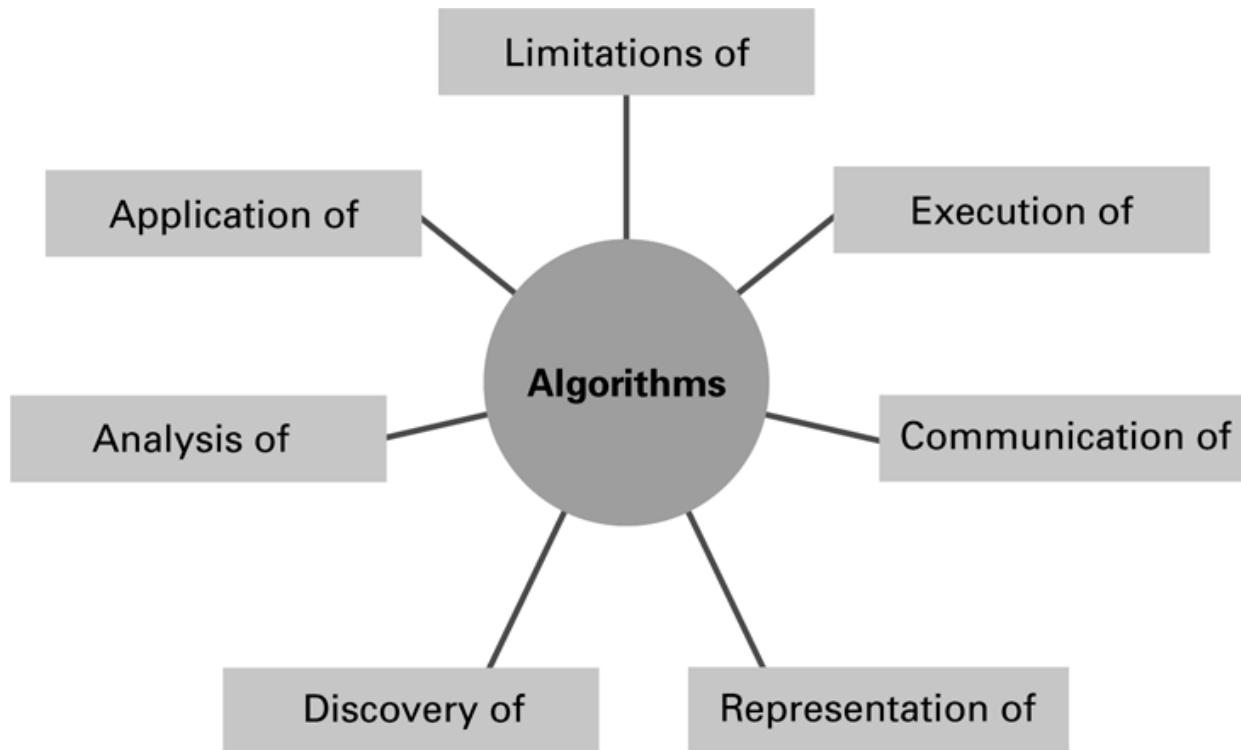
Che cos'è l'informatica?

- Definizione proposta da ACM
(*Association for Computing Machinery*):

*studio sistematico degli algoritmi che
descrivono e trasformano l'informazione:
la loro teoria, analisi, progetto, efficienza,
realizzazione e applicazione*

... è possibile svolgere concettualmente un'attività
di tipo informatico senza l'ausilio del calcolatore!

Che cos'è l'informatica?



“La scienza degli algoritmi”

Il concetto di algoritmo

- Informalmente, un algoritmo è una sequenza di passi, definiti con precisione, che portano alla realizzazione di una determinata operazione.
 - Per esempio, esistono algoritmi per cucinare (ricette), per orientarsi in una città straniera (indicazioni), per eseguire brani musicali (sotto forma di spartiti), etc.

Il concetto di algoritmo

- In ambito matematico, e nelle scienze esatte, il termine algoritmo è usato per indicare un *procedimento di calcolo*, atto alla risoluzione di una determinata classe di problemi.
 - “Algoritmo” deriva dal nome del matematico uzbeko *Muhammad ibn Musa al-Khwarizmi* (vissuto nel IX secolo d.C.), dalla cui opera è nata l'algebra moderna.

Il concetto di algoritmo

- Un algoritmo può essere visto come un *metodo di elaborazione* che, a fronte di un certo input consistente con la natura del problema da risolvere, lo rielabora fino a produrre altri dati come risultato del problema (output).



Algoritmo di Euclide (1/2)

- Algoritmo per calcolare il massimo comune divisore di due interi positivi
 - *Input:* due interi positivi
 - *Output:* massimo comune divisore degli interi dati in input

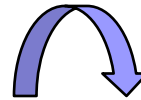
Il massimo comune divisore di due numeri interi a e b che non siano entrambi uguali a zero, $\text{MCD}(a,b)$, è il numero naturale più grande per il quale possono entrambi essere divisi.

Algoritmo di Euclide (2/2)

- *Passo 1:* si assegna ad a e b , rispettivamente, il maggiore e minore dei due valori dati in input
- *Passo 2:* si divide a per b , e si denomina il resto r
- *Passo 3:* se r non è 0, si assegna ad a il valore di b , si assegna a b il valore di r , e si ritorna al Passo 2; altrimenti il massimo comune divisore è il valore assegnato a b

Caratteristiche di un algoritmo

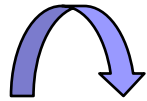
- L'insieme di passi (istruzioni) di un algoritmo deve essere *ordinato*.



L'algoritmo deve avere cioè una struttura ben stabilita in termini di ordine di esecuzione delle istruzioni

Caratteristiche di un algoritmo

- Un algoritmo deve consistere di passi *effettivamente eseguibili*.



Esempio di istruzione
non eseguibile:

“Elencare *tutti* gli interi positivi”

Caratteristiche di un algoritmo

- I passi di un algoritmo devono essere definiti in modo *non ambiguo*.



Il significato di ogni istruzione deve essere univoco per chiunque esegua l'algoritmo

Caratteristiche di un algoritmo

- Un algoritmo definisce un *processo che termina*.
 - Non tutti i problemi possono essere risolti alitmicamente!
 - Il termine algoritmo è spesso usato informalmente anche per riferirsi a sequenze di passi che non hanno necessariamente un termine.



Un esempio è il cosiddetto “algoritmo” della divisione (che non definisce un processo che termina nel caso in cui si divida 1 per 3)

Definizione di algoritmo

- Riassumendo,
possiamo definire un algoritmo come

un procedimento di calcolo/elaborazione

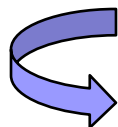
costituito da un insieme ordinato

di passi eseguibili e non ambigui

che giunge certamente a terminazione

Algoritmi e programmi

- È essenziale che un algoritmo sia *comprensibile* al suo esecutore.

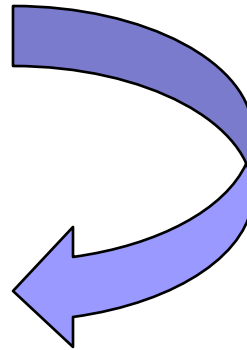


In informatica, gli algoritmi vengono rappresentati tramite *programmi*, cioè sequenze di istruzioni scritte in un opportuno linguaggio, comprensibile al calcolatore.

I linguaggi per la programmazione di algoritmi

Algoritmo

*(linguaggio di progetto/
flow-chart)*



CODIFICA

Programma

*(linguaggio di
programmazione)*

I linguaggi per la programmazione di algoritmi

- Un linguaggio di programmazione è costituito, come ogni altro tipo di linguaggio, da:
 - un insieme di *parole* (il vocabolario) costruite a partire da un insieme di simboli primitivi
 - un insieme di *regole sintattiche* per l'uso corretto delle parole del linguaggio

I linguaggi per la programmazione di algoritmi

- Aspetti coinvolti nella descrizione di un linguaggio:


- Sintassi  *correttezza delle frasi formate a partire dalle parole del linguaggio*

- Semantica  *significato da attribuire alle frasi*


(una frase, pur corretta
dal punto di vista sintattico,
può essere priva di significato!)

I linguaggi per la programmazione di algoritmi

- Notazione BNF (Backus-Naur Form):

< >  *simboli non terminali*

::=  *“è definito come”*

/  *“oppure”*

I linguaggi per la programmazione di algoritmi

- Grammatica formale **$G = (T, N, P, S)$** :

- **T**: insieme dei simboli terminali
cioè delle parole

- **N**: insieme dei simboli non terminali
o *categorie grammaticali* o *costruttori*

I linguaggi per la programmazione di algoritmi

- Grammatica formale **$G = (T, N, P, S)$** :
 - **P**: insieme delle regole di trasformazione o *produzioni* ($A ::= \alpha$, dove A è un simbolo non terminale e α è una concatenazione di simboli terminali e non)
 - **S**: *start symbol* (appartenente all'insieme dei simboli non terminali)

I linguaggi per la programmazione di algoritmi

- L'applicazione delle regole di produzione, partendo dallo *start symbol*, porta alla generazione delle frasi appartenenti al linguaggio.
- Ogni produzione equivale ad una definizione: ciò che deve essere definito appare a sinistra di “ $::=$ ”, mentre la definizione si trova a destra.

I linguaggi per la programmazione di algoritmi

- Formalmente, il linguaggio $L(G)$,
generato da una grammatica G ,

è l'insieme di tutte le frasi formate
esclusivamente da simboli terminali
e ottenibili a partire dallo *start symbol*
attraverso l'applicazione delle
regole di produzione.

I linguaggi per la programmazione di algoritmi

- Esempio:

La frase “*il ragazzo studia*” può essere definita sulla base della seguente “microgrammatica”

- Insieme **T** dei simboli terminali:

{ragazzo, ragazza, il, la, studia, dorme}

- Insieme **N** dei simboli non terminali:

{frase, soggetto, predicato, sostantivo, articolo}

I linguaggi per la programmazione di algoritmi

- Insieme **P** delle regole di produzione espresse in BNF:

< frase > ::= < soggetto > < predicato >

< soggetto > ::= < articolo > < sostantivo >

< articolo > ::= il | la

< sostantivo > ::= ragazzo | ragazza

< predicato > ::= studia | dorme

- Start symbol **S** = frase

I linguaggi per la programmazione di algoritmi

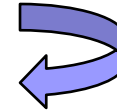


Prospettiva storica

- Agli albori dell'informatica l'uomo, per comunicare i suoi algoritmi al calcolatore, doveva imparare il linguaggio della macchina, costituito da un insieme di istruzioni molto semplici (codificate come stringhe di bit).
 - Linguaggio dipendente dalla specifica macchina
 - Enorme sforzo programmatico richiesto per codificare algoritmi semplici
 - Programmi difficili da scrivere, leggere e mantenere

Prospettiva storica

- La prima evoluzione dei linguaggi di programmazione ha portato ad una codifica di tipo simbolico, anziché binaria, dei programmi.

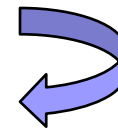


Linguaggi assemblativi o assembly

- Meno “ostici” ma ancora orientati alla macchina
- Traduzione nel linguaggio macchina effettuata da un apposito *programma assemblatore*

Prospettiva storica

- Il passo successivo nell'evoluzione dei linguaggi di programmazione tese a rendere la codifica degli algoritmi il più possibile orientata al problema da risolvere, anziché all'architettura della macchina destinata all'esecuzione del programma.



Linguaggi di alto livello

- Indipendenti dalla macchina hardware sottostante

Linguaggi di alto livello

- Vantaggi fondamentali:
 - I programmatori non devono cimentarsi con i dettagli architetturali di ogni calcolatore.
 - I programmi risultano più semplici da leggere e da modificare.
 - I programmi possono essere eseguiti su qualsiasi macchina, previa traduzione (*portabilità del software*).

Linguaggi di alto livello

- La traduzione nel linguaggio macchina è effettuata da un *compilatore* o da un *interprete*:
 - *compilatore*: opera la traduzione dell'intero programma, prima della sua esecuzione
 - *interprete*: traduce ed esegue il programma, istruzione per istruzione

Linguaggi di alto livello

