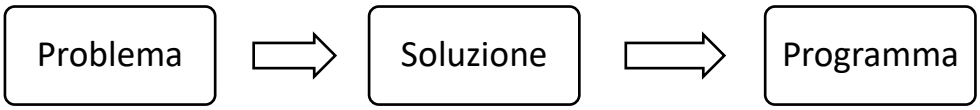
Introduzione

Cos'è un algoritmo?

Un **algoritmo** è una sequenza di passe che prende dei valori in input e restituisce dei valori in output, serve a risolvere dei *problemi computazionali* ben definiti.
Gli algoritmi sono un tassello fondamentale dello sviluppo software.



Creazione di un algoritmo

1) Quale problema devo risolvere? Su quali dati posso contare? Qual è il risultato che mi aspetto?
2) Parte centrale: definisco un procedimento ovvero una sequenza di operazioni che mi porteranno a risolvere il problema
3) Scrivo la soluzione sotto forma di linguaggio di programmazione in modo che possa risultare compreso ed eseguito da un computer.

In primo luogo, è importante sapere quale tipo di problema stiamo andando ad affrontare, interessandoci ad esso nella sua *generalità*, senza concentrarci su valori specifici.
Istanza di input: una specifica configurazione di input in cui i valori rispettano i vincoli del problema.
Una volta che abbiamo un processo esecutivo abbiamo bisogno di un *esecutore* che prenda i dati in input, applichi l'algoritmo e ritorni i valori di output; è importante sapere cosa il nostro esecutore sa fare e ancor di più cosa non sa fare.

Proprietà di un algoritmo

Un algoritmo è composto da una sequenza di istruzioni che devono essere:

- **Ben ordinate:** quando termina un'istruzione devo sapere esattamente quale istruzione verrà dopo, non ci deve essere ambiguità
- **Non ambigue**: l'esecutore deve capire bene il significato di ogni istruzione
- **Realizzabili**: le istruzioni oltre a dover essere comprensibili devono essere effettivamente realizzabili

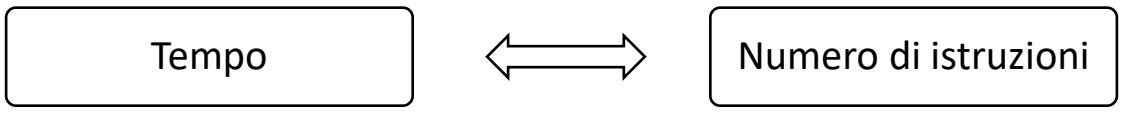
Inoltre, un algoritmo deve:

- **Restituire un risultato** che risolva il nostro problema
- **Terminare**

Algoritmi e caso peggiore

È importante che un algoritmo risulti *corretto,* ovvero per ogni istanza di input deve essere restituita un'istanza di output; per dimostrare che un algoritmo è corretto è necessario avvalersi di una *prova matematica,* non basta dimostrare che un algoritmo è corretto con una sola istanza di input ma basta una sola istanza in cui l'algoritmo non funziona per dimostrare che non è corretto.
Oltre ad essere corretto il nostro algoritmo deve anche essere *efficiente,* ovvero deve utilizzare il minor numero di risorse possibili (anche il tempo è un importante risorsa).

Il concetto di tempo assoluto in cui un algoritmo può operare è relativo all'applicazione su cui è utilizzato: in base alla complessità del problema possiamo tollerare tempi più o meno lunghi.
Per misurare quanto tempo impiega un algoritmo ad essere eseguito conto il numero di operazioni che esegue nel caso peggiore (istanza che richiede il maggior numero di operazioni), senza essere legati alle prestazioni di una macchina specifica, al linguaggio utilizzato, al compilatore…



Perché il caso peggiore?
Il caso peggiore ci da un LIMITE SUPERIORE al numero di operazioni da eseguire per una qualsiasi istanza.

Perché non utilizzare un supercomputer?

Un algoritmo più efficiente eseguito da un computer più lento terminerà comunque SEMPRE prima di uno non efficiente eseguito da un computer più veloce, per un numero infinito di istanze di input.

ESEMPIO

ALGORITMO DI EUCLIDE

- Fino a quando *m* è diverso da *n* esegui le seguenti operazioni:
  - se *m* > *n* assegna ad *m* il valore *(m-n)*
    altrimenti
    assegna *n* il valore *(n - m)*
- Restituisci *n*

Quante operazioni richiede? 3 ad ogni iterazione
Quante iterazioni? Dipende da n ed m…
Qual è il caso peggiore? Che uno dei due numeri sia 1!

Algoritmi efficienti

Non ci interessa esprimere il numero esatto di operazioni ma ci basta una funzione che le esprima in ordine di grandezza.
È una funzione polinomiale? Lineare? Esponenziale?
Il tempo di esecuzione viene espresso in funzione della dimensione dell'input, ovvero quanto è grande la descrizione dell'input

Misure di efficienza

Criterio del costo logaritmico: la dimensione dell'input è il numero di bit necessari per rappresentarlo, si usa quando si ha a che fare con un valore.

Criterio di costo uniforme: la dimensione dell'input è il numero di elementi che lo costituiscono

Altra misura di efficienza è lo **spazio** (quantità di memoria).
N.B.: usare più memoria non sempre è una soluzione accettabile, soprattutto quando si lavora con enormi mole di dati (ad es.: Big Data).

Per quanto riguarda la memoria misuriamo l'occupazione in funzione della dimensione dell'input considerando ancora il caso peggiore

Progettare un algoritmo

Cosa ci serve per progettare un algoritmo?

- Ci serve sapere quali sono le istruzioni che l'esecutore può eseguire
- Ci serve un formalismo che ci permetta di descrivere la sequenza di istruzioni per l'elaboratore
    - **Diagramma di flusso**
    - **Pseudo-Codice**: assomiglia ad un linguaggio di programmazione ad alto livello ma permette di concentrarsi sull'algoritmo rimanendo slegati da una sintassi specifica senza preoccuparsi di problemi a basso livello

Chi è il nostro esecutore? Un **computer.**
Perché? Le macchine sono più veloci degli umani e NON sbagliano (a patto che l'algoritmo sia giusto)

Modello computazionale

MODELLO RAM: astrazione dei computer moderni

- Memoria principale infinita
    - Ogni cella contiene un dato o un'istruzione
    - Tempo di accesso sempre uguale per ogni cella
- Singolo processore
    - In un'unità di tempo può leggere/scrivere o eseguire un'istruzione
    - Istruzioni: operazioni logico aritmetiche, assegnamento, accesso a puntatore

Problemi difficili

Ci sono problemi per i quali non si conosce una soluzione efficiente, questi sono noti come problemi **NP-completi**.
Perché sono interessanti? Sebbene non sia ancora stato trovato un algoritmo efficiente questo non significa che non ne possa esistere uno; in secondo luogo, l'insieme dei problemi NP-completi gode della proprietà che, se esiste un algoritmo efficiente per uno di essi, allora esiste un algoritmo efficiente per ciascuno di essi.
In terzo luogo, molti problemi NP-completi sono simili ma non identici, a problemi per cui conosciamo algoritmi efficienti.
È importante conoscere questi tipi di problemi poiché molti di essi si presentano molto più spesso di quanto ci si possa immaginare e se si incontrano una buona soluzione sarebbe realizzare un algoritmo che fornisca una buona soluzione ma non la migliore possibile.

Strutture Dati

"Insieme di dati indirizzabile con un solo nome organizzati in un modo specifico"

Quando si affronta un problema abbiamo dei dati che vanno organizzati in qualche modo; avere un'organizzazione efficace permette di progettare algoritmi più semplici, eleganti ed efficienti.

Esistono due tipi di strutture dati:

- **Elementari**: dipendono da come i dati vengono memorizzati (array, liste, …)
- **Astratte**: offre una visione ad alto livello dei dati, non ci preoccupiamo di come viene implementata ma ci concentriamo sulle proprietà che devono soddisfare i dati e sulle istruzioni da eseguire (operazioni ammesse)

Una struttura dati può essere implementata in tanti modi diversi, diverse implementazioni possono avere prestazioni diverse.