Capitolo III

Algoritmi e Programmi

Trattamento delle Informazioni

- Informatica = studio sistematico dei *processi* che servono al trattamento delle informazioni o più in generale della definizione della soluzione di problemi assegnati.
  - analisi dettagliata di ciò che serve al trattamento dell'informazione,
  - progetto di una soluzione applicabile alla generazione di informazioni prodotte da altre informazioni,
  - verifica della correttezza e della efficienza della soluzione pensata,
  - manutenzione della soluzione nella fase di funzionamento in esercizio

Informatica e studio di Algoritmi

- “algoritmo”
  - introdotto nella matematica per specificare la sequenza precisa di operazioni il cui svolgimento è necessario per la soluzione di un problema assegnato.
  - Algoritmo & esecutore
- Informatica → studio sistematico degli algoritmi.
  - Il calcolatore è tra tutti gli esecutori di algoritmi (compreso l’uomo) quello che si mostra più potente degli altri e con una potenza tale da permettere di gestire quantità di informazioni altrimenti non trattabili.
- Lo studio dell’Informatica considera quindi il computer come uno scienziato utilizza il proprio microscopio: uno strumento per provare le proprie teorie e, nel caso specifico, verificare i propri ragionamenti o algoritmi.

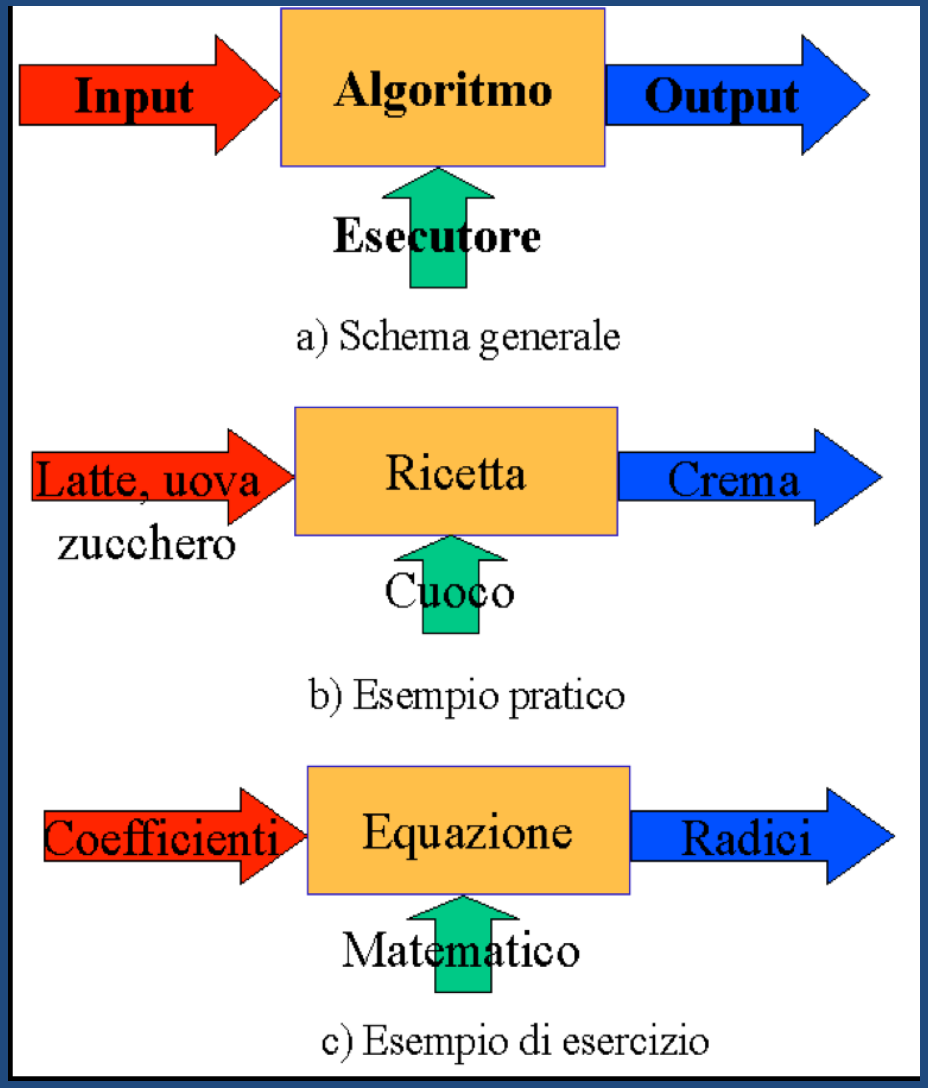
La soluzione dei Problemi: osservazioni

- La descrizione del problema non fornisce, in generale, indicazioni sul metodo risolvente; anzi in alcuni casi presenta imprecisioni e ambiguità che possono portare a soluzioni errate.
  - per alcuni problemi non esiste una soluzione;
  - alcuni problemi, invece, hanno più soluzioni possibili; e quindi bisogna studiare quale tra tutte le soluzioni ammissibili risulta quella più vantaggiosa sulla base di un insieme di parametri prefissati (costo della soluzione, tempi di attuazione, risorse necessarie alla sua realizzazione, etc.)
  - per alcuni problemi non esistono soluzioni eseguibili in tempi ragionevoli e quindi utili.

Alcuni esempi

- **Preparare una torta alla frutta**
  - non si riesce a ricavare alcuna indicazione sulla ricetta da seguire che, tra l'altro, non è facile individuare in un libro di cucina per la sua formulazione generica
- **Risolvere le equazioni di secondo grado**
  - noto e semplice problema di analisi matematica per il quale si conosce chiaramente il procedimento risolvente
- **Individuare il massimo tra tre numeri**
  - esempio di problema impreciso ed ambiguo in quanto non specifica se va cercato il valore massimo o la sua posizione all'interno della terna dei tre numeri assegnati
- **Calcolare le cifre decimali di Π**
  - problema con una soluzione nota che però non arriva a terminazione in quanto le cifre da calcolare sono infinite
- **Inviare un invito ad un insieme di amici**
  - si può osservare che esistono sia soluzioni tradizionali basate sulla posta ordinaria, che soluzioni più moderne quali i messaggi SMS dei telefoni cellulari o i messaggi di posta elettronica
    - Bisogna quindi scegliere la soluzione più conveniente: ad esempio quella che presenta un costo più basso
- **Individuare le tracce del passaggio di extraterrestri**
  - chiaro esempio di problema che non ammette soluzione: o, come si dice, non risolvibile

Schema di soluzione



Algoritmo ed Esecutore

- algoritmo
  - un testo che prescrive un insieme di operazioni od azioni eseguendo le quali è possibile risolvere il problema assegnato
  - Se si indica con istruzione la prescrizione di una singola operazione, allora l'algoritmo è un insieme di istruzioni da svolgere secondo un ordine prefissato
- esecutore
  - l'uomo o la macchina in grado di risolvere il problema eseguendo l'algoritmo.
  - Se un algoritmo è un insieme di istruzioni da eseguire secondo un ordine prefissato, allora l'esecutore non solo deve comprendere le singole istruzioni ma deve essere anche capace di eseguirle.
    - una dopo l'altra secondo un ordine rigidamente **sequenziale** che impone l'inizio dell'esecuzione di una nuova istruzione solo al termine di quella precedente, oppure più istruzioni contemporaneamente svolgendole in **parallelo**;
- informazioni di ingresso (anche dette input)
  - le informazioni che devono essere fornite affinché avvengano le trasformazioni desiderate;
- informazioni di uscita (anche dette output)
  - i risultati prodotti dall'esecutore del dato algoritmo.

Concetto di Automa

- Elaborazione → concetto matematico di funzione

  Y=F(X)

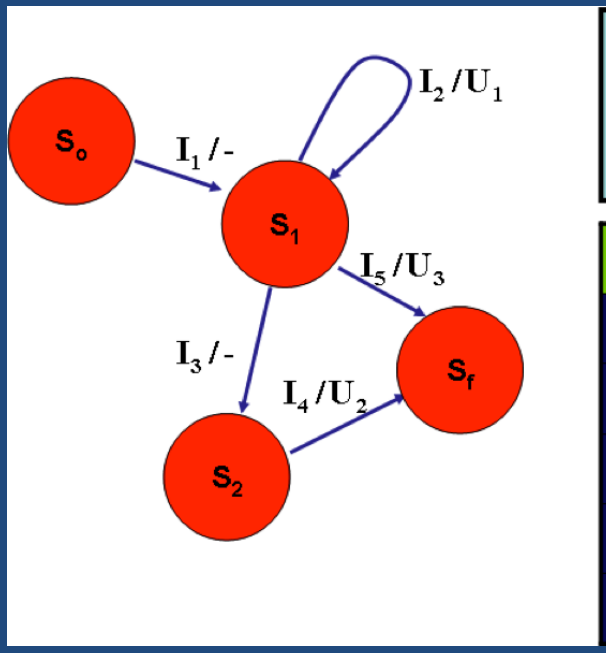
  in cui X sono i dati iniziali da elaborare, Y i dati finali o risultati e F è la regola di trasformazione.
- Automa a stati finiti → astrazione per il concetto di **macchina che esegue algoritmi**
- **concetto di stato**
  - particolare condizione di funzionamento in cui può trovarsi la macchina.
- **L'automa è uno dei modelli fondamentali dell'informatica**
  - E' applicabile a qualsiasi sistema che evolve nel tempo per effetto di sollecitazioni esterne.
  - Ogni sistema se soggetto a sollecitazioni in ingresso risponde in funzione della sua situazione attuale eventualmente emettendo dei segnali di uscita, l'effetto della sollecitazione in ingresso è il mutamento dello stato del sistema stesso.
  - Il sistema ha sempre uno stato iniziale di partenza da cui inizia la sua evoluzione.
  - Può terminare in uno stato finale dopo aver attraversato una serie di stati intermedi.

Automa a Stati Finiti

- Un automa M (a stati finiti) può essere definito da una quintupla di elementi (Q,I,U,t,w) dove:
  - Q è un insieme finito di stati interni caratterizzanti l'evoluzione del sistema;
  - I è un insieme finito di sollecitazioni in ingresso;
  - U è un insieme finito di uscite;
  - t è la funzione di transizione che trasforma il prodotto cartesiano Q×I in Q (t: QxI → Q)
  - w è la funzione di uscita che trasforma Q×I in U (w: QxI → U).

Rappresentazione a grafo

- Grafo
  - un cerchio per rappresentare gli stati del sistema
  - archi orientati ad indicare le transizioni



$Q = (S_o, S_1, S_2, S_f)$
$I = (I_1, I_2, I_3, I_4, I_5)$
$U = (U_1, U_2, U_3)$

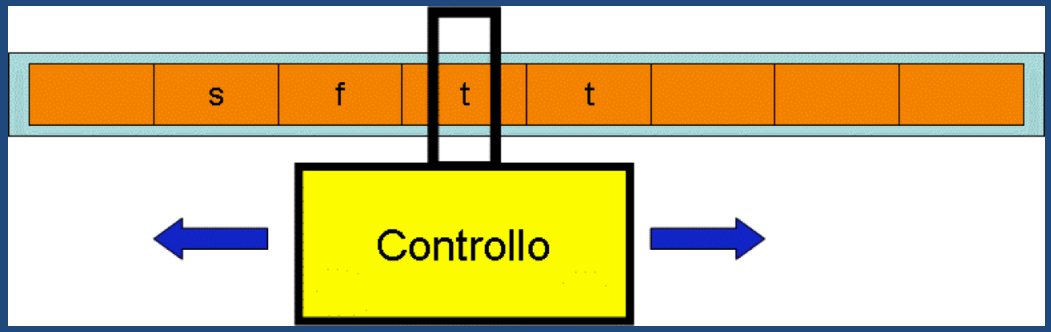
I / S	S_o	S_1	S_2	S_f
I_1	S_1 /-			
I_2		S_1 /U_1		
I_3		S_2 /-		
I_4			S_f /U_2	
I_5		S_f /U_3		

Il Modello di Macchina di Turing

- Il modello di Macchina di Turing è un particolare automa per il quale sono definiti
  - l'insieme degli ingressi e delle uscite come insiemi di simboli
  - è definito un particolare meccanismo di lettura e scrittura delle informazioni.
- È un modello fondamentale nella teoria dell'informatica, in quanto permette di raggiungere risultati teorici sulla calcolabilità e sulla complessità degli algoritmi.

Macchina di Turing

- E’ un automa con testina di scrittura/lettura su nastro bidirezionale potenzialmente illimitato.
  - Ad ogni istante la macchina si trova in uno stato appartenente ad un insieme finito e legge un simbolo sul nastro.
  - La funzione di transizione, in modo deterministico
    - fa scrivere un simbolo
    - fa spostare la testina in una direzione o nell'altra
    - fa cambiare lo stato.
- La macchina si compone di:
  - una memoria costituita da un nastro di dimensione infinita diviso in celle; ogni cella contiene un simbolo oppure è vuota;
  - una testina di lettura scrittura posizionabile sulle celle del nastro;
  - un dispositivo di controllo che, per ogni coppia (stato, simbolo letto) determina il cambiamento di stato ed esegue un’azione elaborativa.



Definizione Formale

- è definita dalla quintupla:

$$(A, S, f_m , f_s, f_d)$$

  - A è l'insieme finito dei simboli di ingresso e uscita;
  - S è l'insieme finito degli stati (di cui uno è quello di terminazione);
  - f_m è la funzione di macchina definita come $A \times S \rightarrow A$;
  - f_s è la funzione di stato $A \times S \rightarrow S$;
  - f_d è la funzione di direzione $A \times S \rightarrow D = \{Sinistra, Destra, Nessuna\}$
- La macchina è capace di:
  - leggere un simbolo dal nastro;
  - scrivere sul nastro il simbolo specificato dalla funzione di macchina;
  - transitare in un nuovo stato interno specificato dalla funzione di stato;
  - spostarsi sul nastro di una posizione nella direzione indicata dalla funzione di direzione.
- **La macchina si ferma quando raggiunge lo stato di terminazione**

Esempio di MdT

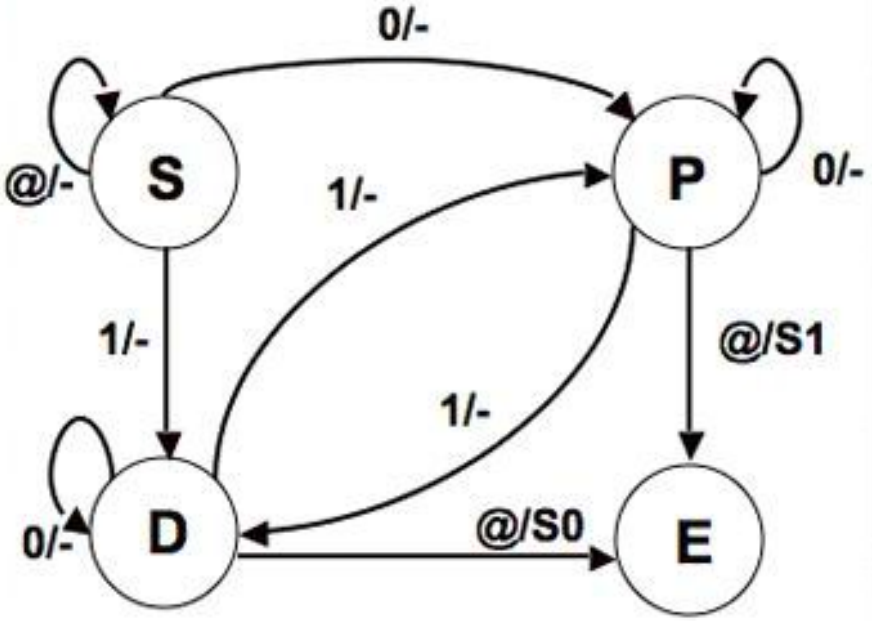
- Costruiamo una macchina che valuti se il numero di occorrenze del simbolo 1 in una sequenza di 0 e 1 è pari.
  - Consideriamo in aggiunta che la sequenza debba iniziare e terminare con un simbolo speciale, ad esempio **@.**
  - **Rappresentazione della sequenza: @0101110@**
  - **Rappresentazione del risultato:**
    - **Numero di 1 pari, la macchina scrive 1 nella prima casella a destra della stringa di ingresso**
    - **Numero di 1 dispari, la macchina scrive 0 nella prima casella a destra della stringa di ingresso**

Progetto del Controllo

- Il dispositivo di Controllo è un automa:
  - **Stati:**
    - **Stato S: stato iniziale in cui la macchina resta fino a quando non individua la stringa di ingresso sul nastro.**
    - **Stato P: il numero di 1 esaminato è pari.**
    - **Stato D: il numero di 1 esaminato è dispari.**
    - **Stato E: la macchina ha eseguito l'algoritmo e si ferma.**
    - **Ingressi1, 0, @**
    - **Uscite**
      - **Spostamento a destra (>), spostamento a sinistra (<), resta fermo (f), scrivi 1 (S1), scrivi 0 (S0)**

Unità di controllo

Quintuple				
$S_{iniziale}$	Leggo	Scrivo	Azione	S_{finale}
S	@	-	>	S
S	1	-	>	D
S	0	-	>	P
P	@	1	f	E
P	1	-	>	D
P	0	-	>	P
D	@	0	f	E
D	0	-	>	D
D	1	-	>	P
E	-	-	f	E



Macchina di Turing e Algoritmi

- Una macchina di Turing che
  - si arresti
  - trasformi un nastro t in uno t'

  rappresenta l'algoritmo per l'elaborazione Y=F(X), ove X e Y sono codificati rispettivamente in t e t'.

- Una macchina di Turing la cui parte di controllo è capace di leggere da un nastro anche **la descrizione dell'algoritmo** è **una macchina universale** capace di simulare il lavoro compiuto da un'altra macchina qualsiasi.
  - leggere dal nastro la descrizione dell'algoritmo richiede di saper interpretare il linguaggio con il quale esso è stato descritto.

- La **Macchina di Turing Universale** è l'**interprete** di un linguaggio

Tesi di Church e Turing

- **Tesi di Church e Turing**: ***Non esiste alcun formalismo, per modellare una determinata computazione meccanica, che sia più potente della Macchina di Turing e dei formalismi ad essi equivalenti***.
  - Ogni algoritmo può essere codificato in termini di Macchina di Turing ed è quindi ciò che può essere eseguito da una macchina di Turing.
  - Un problema è non risolubile algoritmicamente se nessuna Macchina di Turing è in grado di fornire la soluzione al problema in tempo finito.
  - Se dunque esistono problemi che la macchina di Turing non può risolvere, si conclude che esistono algoritmi che non possono essere **calcolati**.
- Sono problemi **decidibili** quei problemi che possono essere meccanicamente risolvibili da una macchina di Turing; sono indecidibili tutti gli altri.

Conseguenze

- se per problema esiste un algoritmo risolvente **questo è indipendente dal sistema che lo esegue**
  - se è vero che per esso esiste una macchina di Turing;
- **l'algoritmo è indipendente** dal linguaggio usato per descriverlo
  - visto che per ogni linguaggio si può sempre definire una macchina di Turing universale.

Calcolabilità

- La teoria della calcolabilità cerca di comprendere quali funzioni ammettono un procedimento di calcolo automatico.
- Nella teoria della calcolabilità la tesi di Church-Turing è un'ipotesi che intuitivamente dice che se un problema si può calcolare, allora esisterà una macchina di Turing (o un dispositivo equivalente, come il computer) in grado di risolverlo (cioè di calcolarlo).
- **Più formalmente possiamo dire che la classe delle funzioni calcolabili coincide con quella delle funzioni calcolabili da una macchina di Turing.**

MdT e Von Neumann

- La macchina di Turing e la macchina di von Neumann sono due modelli di calcolo fondamentali per caratterizzare la modalità di descrizione e di esecuzione degli algoritmi.
- La macchina di Von Neumann fu modellata dalla Macchina di Turing Universale per ciò che attiene alle sue modalità di computazione.
  - La sua memoria è però **limitata** a differenza del nastro di Turing che ha lunghezza infinita.
  - La macchina di Von Neumann, come modello di riferimento per sistemi non solo teorici, prevede anche la dimensione dell'interazione attraverso i suoi dispositivi di input ed output.

Trattabilità

- calcolabilità
  - consente di dimostrare *l'esistenza di un algoritmo* risolvente un problema assegnato ed indipendente da qualsiasi automa
- trattabilità
  - studia la eseguibilità di un algoritmo da parte di un sistema informatico.
- Esistono problemi classificati come risolvibili ma praticamente intrattabili non solo dagli attuali elaboratori ma anche da quelli, sicuramente più potenti, del futuro.
- Sono noti problemi che
  - pur presentando un algoritmo di soluzione, non consentono di produrre risultati in tempi ragionevoli neppure se eseguiti dal calcolatore più veloce.
  - ammettono soluzione di per sé lenta e quindi inaccettabile.

Trattabilità e Complessità

- Il concetto di trattabilità è legato a quello di complessità computazionale
  - .. studia i costi intrinseci alla soluzione dei problemi, con l'obiettivo di comprendere le prestazioni massime raggiungibili da un algoritmo applicato a un problema.
- La complessità consente di individuare i problemi risolvibili che siano trattabili da un elaboratore con costi di risoluzione che crescano in modo ragionevole al crescere della dimensione del problema.
- Tali problemi vengono detti trattabili

Spazio e Tempo

- La complessità di un algoritmo corrisponde a una misura delle risorse di calcolo consumate durante la computazione ed è tanto più elevata quanto maggiori sono le risorse consumate.
- Complessità spaziale
  - …. quantità di memoria necessaria alla rappresentazione dei dati necessari all'algoritmo per risolvere il problema;
- Complessità temporale
  - …. il tempo richiesto per produrre la soluzione.

- Le misure di complessità possono essere:
  - statiche
    - se sono basate sulle caratteristiche strutturali (ad esempio il numero di istruzioni) dell'algoritmo e prescindono dai dati di input su cui esso opera.
  - dinamiche
    - se tengono conto sia delle caratteristiche strutturali dell'algoritmo che dei dati di input su cui esso opera

Complessità e dati di input

- Un primo fattore che incide sul tempo impiegato dall'algoritmo è la quantità di dati su cui l'algoritmo deve lavorare
  - il tempo di esecuzione è solitamente espresso come una funzione f(n) della dimensione n dei dati di input.
    - Si dirà, ad esempio, che un algoritmo ha un tempo di esecuzione n^2 se il tempo impiegato è pari al quadrato della dimensione dell'input.
- Da sola la dimensione dei dati di input non basta.
- Il tempo di esecuzione dipende anche dalla configurazione dei dati in input oltre che dalla loro dimensione

Tempo di Esecuzione e configurazione dei dati in ingresso

- analisi del caso migliore
  - … per calcolare il tempo di esecuzione quando la configurazione dei dati presenta difficoltà minime di trattamento.
- analisi del caso peggiore
  - …. per calcolare il tempo di esecuzione quando la configurazione dei dati presenta difficoltà massime di trattamento.
    - Si tratta di un'analisi molto utile, perché fornisce delle garanzie sul tempo massimo che l'algoritmo può impiegare
- analisi del caso medio
  - per calcolare il tempo di esecuzione quando la configurazione presenta difficoltà medie di trattamento

Complessità Asintotica

- Interessa sapere come l'ordine di grandezza del tempo di esecuzione cresce al limite, ossia per dimensioni dell'input sufficientemente grandi quando la funzione *f(n)* tende ai suoi asintoti.
  - Lo studio della condizione asintotica della complessità di un algoritmo permette ulteriormente di trascurare in un algoritmo operazioni non significative concentrando l'attenzione solo su quelle predominanti.
  - dati due algoritmi diversi che risolvono lo stesso problema e presentano due diverse complessità f(n) e g(n), se f(n) è asintoticamente inferiore a g(n) allora esiste una dimensione dell'input oltre la quale l'ordine di grandezza del tempo di esecuzione del primo algoritmo è inferiore all'ordine di grandezza del tempo di esecuzione del secondo.
- La complessità asintotica dipende solo dall'algoritmo, mentre la complessità esatta dipende da tanti fattori legati alla esecuzione dell'algoritmo

Tipi di Complessità

- A seconda della funzione f(n) asintotica, si possono individuare:
- complessità di tipo polinomiale
  - n (lineare)
  - n^2, n^3, $n^{5, \ldots}$
- Complessità di tipo esponenziale (o, in generale, Non Polinomiale, NP)
  - 2^n, 3^n , ….

	10	20	30	40	50
n	0,00001 sec	0,00002 sec	0,00003 sec	0,00004 sec	0,00005 sec
n^2	0,0001 sec	0,0004 sec	0,0009 sec	0,0016 sec	0,0025 sec
n^3	0,001 sec	0,008 sec	0,027 sec	0,064 sec	0,125 sec
n^5	0,1 sec	3,2 sec	24,3 sec	1,7 min	5,2 min
2^n	0,001 sec	1,0 sec	17,9 min	12,7 giorni	35,7 anni
3^n	0,059 sec	58 min	6,5 anni	3,855 secoli	200.000.000 secoli

*.. Per un elaboratore capace di eseguire un milione istruzioni al secondo (1 MIPS).

La descrizione degli algoritmi

- Dato un problema …
  - … capire preliminarmente se il problema ammette soluzioni
  - … nel caso ne ammetta, individuare un metodo risolutivo (algoritmo)
  - …. esprimere tale metodo in un linguaggio comprensibile all'esecutore a cui è rivolto

Un esempio


La comne osservazione dela evoluzione del mondo degli elaboratori eletronici non deve trarre in ingano. Difati a dispeto del'evoluzione tecnologicha, l'attivita di programmazione di propone come un'arte che è ben vero che muta, ma che non può non contenere i trati antropomorfi di chi genera i programmi e di chi li utilizza.
La comune osservazione della evoluzione del mondo degli elaboratori elettronici non deve trarre in inganno. Difatti a dispetto dell'evoluzione tecnologica, l'attività di programmazione di propone come un'arte che è ben vero che muta, ma che non può non contenere i tratti antropomorfi di chi genera i programmi e di chi li utilizza.

Soluzione …

```
1       leggi un rigo del testo;
2       per ogni parola del rigo
        fai:
3               se conosci la parola,
                allora controlla come è scritta
                altrimenti fai  una ricerca nel vocabolario
4               se la parola non è riportata in modo corretto
                allora correggila,
5               riscrivi la parola nel testo corretto;
6       ripeti le azioni da 1) a 5)
        fino alla terminazione dei righi del testo.
```

Strutture di Controllo

- Si usano naturalmente *costrutti* che fissano l'ordine in cui le diverse azioni devono essere svolte.
  - Il più semplice tra essi è quello che stabilisce che le azioni devono essere svolte una dopo l'altra. (sequenza)
  - un altro costrutto stabilisce che alcune azioni devono essere svolte solo se si verificano determinate condizioni (selezione)
    - la frase "se la parola è sbagliata, allora correggi, altrimenti non fare niente" prescrive la correzione soltanto in presenza di un errore.
  - un ultimo costrutto dice che alcune azioni devono essere ripetute un numero di volte prestabilito
    - ("per ogni parola del rigo fai")

    o determinato dal verificarsi di certe condizioni (iterazione)
    - ("ripeti le azioni da 1) a 4) fino alla terminazione del testo")

Tipi di Istruzione

- *Istruzioni:*
  - *elementari* quelle istruzioni che l'esecutore è in grado di comprendere ed eseguire;
  - *non elementari* quelle non note all'esecutore.
- Perché un'istruzione non elementare possa essere eseguita dall'esecutore a cui è rivolta, deve essere specificata in termini più semplici.
- Il procedimento che trasforma una istruzione non elementare in un insieme di istruzioni elementari, prende il nome di *raffinamento o specificazione dell'istruzione non elementare*.
  - Il processo di raffinamento è molto importante. Senza di esso si dovrebbero esprimere gli algoritmi direttamente nel linguaggio di programmazione disponibile

Caratteristiche delle operazioni di un algoritmo

- finitezza:
  - Le operazioni devono avere termine entro un intervallo di tempo finito dall'inizio della loro esecuzione;
- descrivibilità:
  - Le operazioni devono produrre, se eseguite, degli effetti descrivibili, per esempio fotografando lo stato degli oggetti coinvolti sia prima che dopo l'esecuzione dell'operazione;
- riproducibilità:
  - Le operazioni devono produrre lo stesso effetto ogni volta che vengono eseguite nelle stesse condizioni iniziali;
- comprensibilità:
  - Le operazioni devono essere espresse in una forma comprensibile all'esecutore che deve eseguirle.

Sequenza Dinamica

- L'esecuzione di un algoritmo da parte di un esecutore si traduce in una successione di azioni che vengono effettuate nel tempo.
  - Si dice che l'esecuzione di un algoritmo evoca un processo sequenziale, cioè una serie di eventi che occorrono uno dopo l'altro, ciascuno con un inizio e una fine identificabili.
  - Si definisce *sequenza di esecuzione* la descrizione del processo sequenziale. La sequenza di esecuzione è l'elenco di tutte le istruzioni eseguite, nell'ordine di esecuzione, e per questo motivo viene anche detta sequenza dinamica.

Dipendenza della sequenza dall'input

Calcolo $d = \sqrt{b^2 - 4ac}$

Calcolo la prima radice come $x_1 = \dfrac{-b + d}{2a}$

Calcolo la seconda radice come $x_2 = \dfrac{-b - d}{2a}$

Calcolo $\Delta = b^2 - 4ac$

Se $\Delta \geq 0$ allora

    Calcolo $d = \sqrt{\Delta}$

    Calcolo la prima radice come $x_1 = \dfrac{-b + d}{2a}$

    Calcolo la seconda radice come $x_2 = \dfrac{-b - d}{2a}$

Altrimenti

    Calcola radici complesse

… 2 sequenze

Calcolo $\Delta = b^2 - 4ac$

Verifico che $\Delta \geq 0$ *è risultata vera*

Calcolo $d = \sqrt{\Delta}$

Calcolo la prima radice come $x_1 = \dfrac{-b + d}{2a}$

Calcolo la seconda radice come $x_2 = \dfrac{-b - d}{2a}$

Calcolo $\Delta = b^2 - 4ac$

Verifico che $\Delta \geq 0$ *è risultata non vera*

Calcolo radici complesse

Il solitario del carcerato: esempio

Fintantoché (il mazzo ha ancora carte) ripeti:
    Mischia le carte
    Prendi 4 carte dal mazzo e disponile sul tavolo di gioco
    Se (le 4 carte hanno la stessa figura) allora levale dal tavolo di gioco

Mischiate le carte
Prese le 4 carte dal mazzo e messe sul tavolo di gioco
Verificato che (le 4 carte hanno la stessa figura) è risultata vera
Tolte le 4 carte dal tavolo di gioco

Mischiate le carte
Prese le 4 carte dal mazzo e messe sul tavolo di gioco
Verificato che (le 4 carte hanno la stessa figura) *è risultata falsa*
Mischiate le carte
Prese le 4 carte dal mazzo e messe sul tavolo di gioco
Verificato che (le 4 carte hanno la stessa figura) *è risultata vera*
Tolte le 4 carte dal tavolo di gioco

… solitario … sequenze

ripetuto n-1 volte
    Mischiate le carte
    Prese le 4 carte dal mazzo e messe sul tavolo di gioco
    Verificato che (le 4 carte hanno la stessa figura) *è risultata falsa*
Mischiate le carte
Prese le 4 carte dal mazzo e messe sul tavolo di gioco
Verificato che (le 4 carte hanno la stessa figura) *è risultata vera*
Tolte le 4 carte dal tavolo di gioco

ripetuto infinite volte
    Mischiate le carte
    Prese le 4 carte dal mazzo e messe sul tavolo di gioco
    Verificato che (le 4 carte hanno la stessa figura) *è risultata falsa*

- Un algoritmo può prescrivere più di una sequenza di esecuzione.
- Se poi l'algoritmo prescrive un processo ciclico, può accadere che il numero di sequenze sia infinito. In questo caso l'algoritmo prescrive un processo che non ha mai termine.

Sequenza Statica e Dinamica

- In un programma
  - … descrizione dell'algoritmo in un linguaggio di programmazione

  la sequenza lessicografica (anche chiamata *sequenza statica*) delle istruzioni
  - …. descrizione delle azioni da svolgere nel linguaggio di programmazione

  descrive una *pluralità di sequenze dinamiche* differenti.
- Il numero di sequenze dinamiche non è noto a priori e dipende dai dati da elaborare.
- La valutazione sia del tipo che del numero delle sequenze dinamiche è di fondamentale importanza per valutare soluzioni diverse dello stesso problema in modo
  - da poter dire quale di esse presenta un tempo di esecuzione migliore
  - da poter affermare se una soluzione ha terminazione o meno.

Linguaggio di Programmazione

- Il linguaggio di programmazione è una **notazione formale** per descrivere algoritmi e, come ogni linguaggio, è dotato di un **alfabeto, un lessico, una sintassi ed una semantica.**
- L'aspetto formale del linguaggio si manifesta soprattutto nella presenza di **regole rigide** per la composizione di programmi a partire dai semplici caratteri dell'alfabeto.
  - L'insieme di queste regole costituisce la **grammatica del linguaggio**.
- Un limitato insieme di regole definisce la **struttura lessicale** del programma
  - .. o unità elementari, cioè le parole del linguaggio.
  - Il lessico, quindi, permette di strutturare l'insieme limitato dei caratteri dell'alfabeto nel vocabolario.
- L'organizzazione delle parole in frasi è invece guidata da regole che compongono la **sintassi del linguaggio**.
- Infine l'attribuzione di un significato alle frasi è oggetto delle **regole semantiche**.

Dal Linguaggio Macchina …

- Il linguaggio *più semplice* è quello che è direttamente compreso dalla CPU.
  - Chiameremo ***linguaggio macchina*** tale linguaggio per evidenziare il suo stretto legame con l'hardware.
- E’ difficile programmare in linguaggio macchina
  - … gran numero di comandi per singole istruzioni
  - …. istruzioni espresse sotto forma di sequenze di
- I **linguaggi assemblativi**
  - … pur mantenendo uno stretto legame con le potenzialità offerte dal linguaggio macchina,

  sostituiscono alle sequenze di bit dei **codici mnemonici** più facili da interpretare e ricordare.
- I linguaggi macchina e assemblativi sono anche comunemente detti linguaggi di basso livello, in quanto si pongono al livello della macchina

… ai linguaggi ad alto livello

- **Linguaggi ad Alto Livello**
  - linguaggi di programmazione che, con istruzioni più sintetiche
    - *ossia con istruzioni più vicine al tradizionale modo di esprimere i procedimenti di calcolo da parte di un essere umano,*
  - rendono l'attività di programmazione più semplice;
  - Fanno uso di uno *pseudo-linguaggio umano,* utilizzando per il loro scopo *parole-chiave* o *codici operativi* ispirati quasi esclusivamente alla lingua inglese.

Metalinguaggi

- …. linguaggi con i quali si presentano **le regole della grammatica di un qualsiasi linguaggio**.
  - Un metalinguaggio descrive quindi le proprietà sintattiche e semantiche relative a un altro linguaggio o al linguaggio stesso.
- BNF, Backus Naur Normal Form
  - Metalinguaggio sviluppato da John Backus e Peter Naur nel 1960
  - Prescrive che le frasi, che compongono un linguaggio, possono essere specificate a partire da un unico simbolo iniziale di alto livello.
  - Ogni simbolo di alto livello (non terminale) può essere sostituito da una frase fra un insieme di sottofrasi, comprendenti simboli sia non terminali che di basso livello (terminali), mentre i simboli terminali non possono essere sostituiti.
  - Il processo si arresta quando si ottiene una sequenza di soli simboli terminali.

BNF - caratteristiche

- Il linguaggio definito da una BNF è costituito da tutte le sequenze di simboli terminali ottenute applicando questo processo a partire dal simbolo iniziale di alto livello.
  - I simboli non terminali sono racchiusi fra parentesi angolari (<),
  - i terminali fra virgolette ("),
  - le diverse alternative per un non terminale sono separate da una barra verticale (|)
  - e il segno := indica le possibili sostituzioni per il non terminale.
  - Esempio:
    - <Frase> := <Soggetto> <Predicato verbale> <Complemento oggetto>
    - <Soggetto> := “io” | “tu” | “mario”
    - <Predicato verbale> := "mangia" | “beve”
    - <Complemento oggetto> := “mela” | “l’acqua”

EBNF

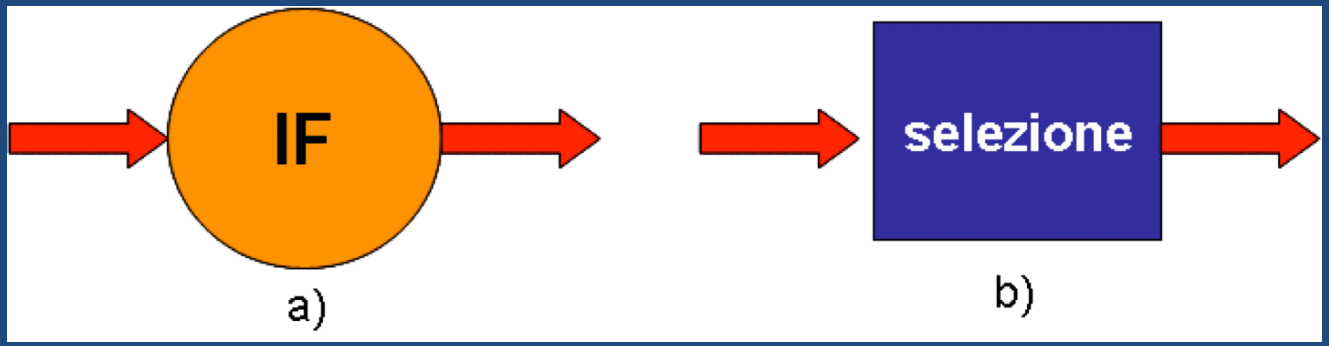
- Una estensione della BNF, detta Extendend BNF (EBNF), è stata introdotta da Niklaus Wirth per specificare la sintassi del linguaggio di programmazione Pascal.
- Nella EBNF i simboli {,}, (,) e * possono comparire nel lato destro di una frase.
  - Una sequenza di simboli racchiusa fra { e } indica che tale sequenza è opzionale,
  - … mentre un simbolo, o una sequenza racchiusa fra (e), seguito da *, indica un numero arbitrario di ripetizioni della sequenza, incluso 0.

Carte Sintattiche

- Sono strumento di tipo grafico usate per la descrizione delle regole del linguaggio.
  - Una carta sintattica è un grafo in cui linee orientate uniscono delle scatole di formato differente: seguendo le frecce ed interpretando opportunamente il contenuto delle scatole si costruiscono frasi sintatticamente corrette.

Simboli terminali e non

- Le scatole rettangolari in figura sono dette SIMBOLI NON TERMINALI in quanto il loro contenuto non compare direttamente nelle frasi.
  - Esse fanno riferimento ad altre carte sintattiche (quelle il cui nome coincide col contenuto della scatola) che virtualmente devono essere inserite nella posizione assunta dal nodo del grafo.
- Le scatole tonde della figura sono dette SIMBOLI TERMINALI in quanto contengono caratteri o sequenze di caratteri che devono apparire direttamente nella frase, nella posizione assunta dal nodo del grafo.



Caratteristiche delle Carte Sintattiche

- Le carte sintattiche non solo permettono un rapido apprendimento delle regole della grammatica del linguaggio grazie all'immediatezza fornita dalla rappresentazione di natura grafica, ma mettono anche nella condizione di scoprire facilmente eventuali errori presenti nella frase.
  - Difatti così come data una carta si è in grado di costruire una frase corretta del linguaggio, si può partendo dalla frase controllarne la correttezza ripercorrendo la carta corrispondente ed in caso negativo anche individuare la causa degli errori

Programmazione Strutturata

- Il ruolo degli algoritmi è fondamentale se si pensa che **essi sono indipendenti sia dal linguaggio** in cui sono espressi sia dal **computer che li esegue**.
- Si pensi ad una ricetta per una torta alla frutta:
  - il procedimento è lo stesso indipendentemente dall'idioma usato;
  - la torta prodotta (dovrebbe ☺ …) è la stessa indipendentemente dal cuoco.

La progettazione degli algoritmi

- Il progetto degli algoritmi è una onerosa attività intellettuale
  - …. molto più onerosa di quella di esprimere l'algoritmo con un linguaggio di programmazione

  che richiede **creatività ed intuito**.
- Non esiste un algoritmo per il progetto degli algoritmi!
- Valutazione della complessità della soluzione individuata:
  - se ci sono algoritmi diversi per descrivere lo stesso processo:
    - la *complessità* ci dice quale di essi è migliore, ossia quale viene eseguito nel minor tempo con la minima occupazione di memoria, più in generale con il miglior utilizzo di tutte le risorse disponibili.
    - lo studio della *correttezza* ci consente di valutare l'aderenza della soluzione alle specifiche del problema.
      - Essere sicuri della correttezza è un'attività tanto più complessa quanto più complesso risulta il progetto dell'algoritmo.

L’Ingegneria del Software

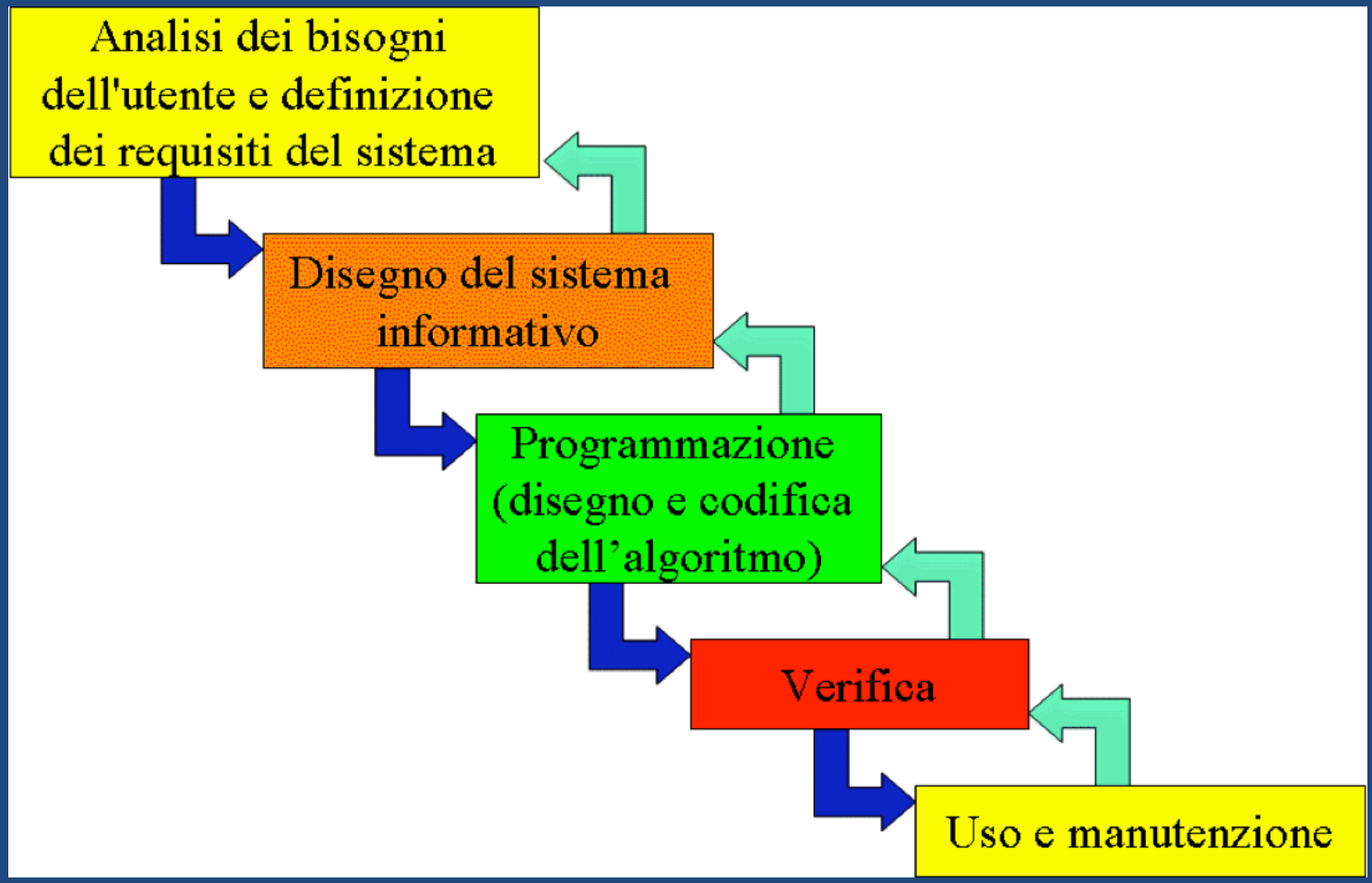
- Ai fini di una produzione industriale di qualità è necessario evitare di produrre software in base alle esperienze e/o alle iniziative del programmatore:
  - il processo di produzione del software non può essere un processo di tipo artigianale
    - Ad esempio, negli anni '60 il programmatore usava mille trucchi per risparmiare memoria!

    deve invece servirsi di **metodologie e tecniche sistematiche** di progettazione e programmazione con fissati parametri di qualità e in maniera standard.
- La "***software engineering***" (ingegneria del software) è la branca dell'Ingegneria Informatica che raccoglie il patrimonio di metodi e tecniche per la produzione del software.

Requisiti del Software

- buon livello qualitativo;
- produttività medio-alta;
- impiego di personale non troppo specializzato (la specializzazione é fornita dallo standard);
- riduzione sensibile del costo del prodotto.

Il ciclo di vita del software



La programmazione Strutturata

- L'obiettivo principale della programmazione strutturata consiste nella costruzione di programmi con le seguenti caratteristiche:
  - **leggibilità**: un programma deve essere comprensibile ad altri programmatori;
  - **documentabilità**: un programma deve contenere al suo interno il significato e la motivazione di tutte le scelte di progetto effettuate;
  - **modificabilità**: un programma deve essere strutturato in modo da permettere un rapido adattamento ad una eventuale variazione di alcuni parametri del progetto;
  - **provabilità**: un programma deve essere costruito in modo da facilitare le attività di testing e debugging (controllo della correttezza e correzione degli errori) fin dalle prime fasi del progetto software.

Caratteristiche: modularità

- La modularizzazione ci dice che un programma deve essere composto di moduli funzionali.
  - Ogni modulo funzionale deve possedere un singolo e ben precisato compito.
  - Ogni modulo deve essere dotato di un solo ingresso e di una sola uscita.
- La modularizzazione comporta una regolarità della struttura dei programmi ed un valido supporto per la fase di progetto, in quanto rispecchia la limitazione degli esseri umani ad esaminare *un solo aspetto di un problema alla volta*.

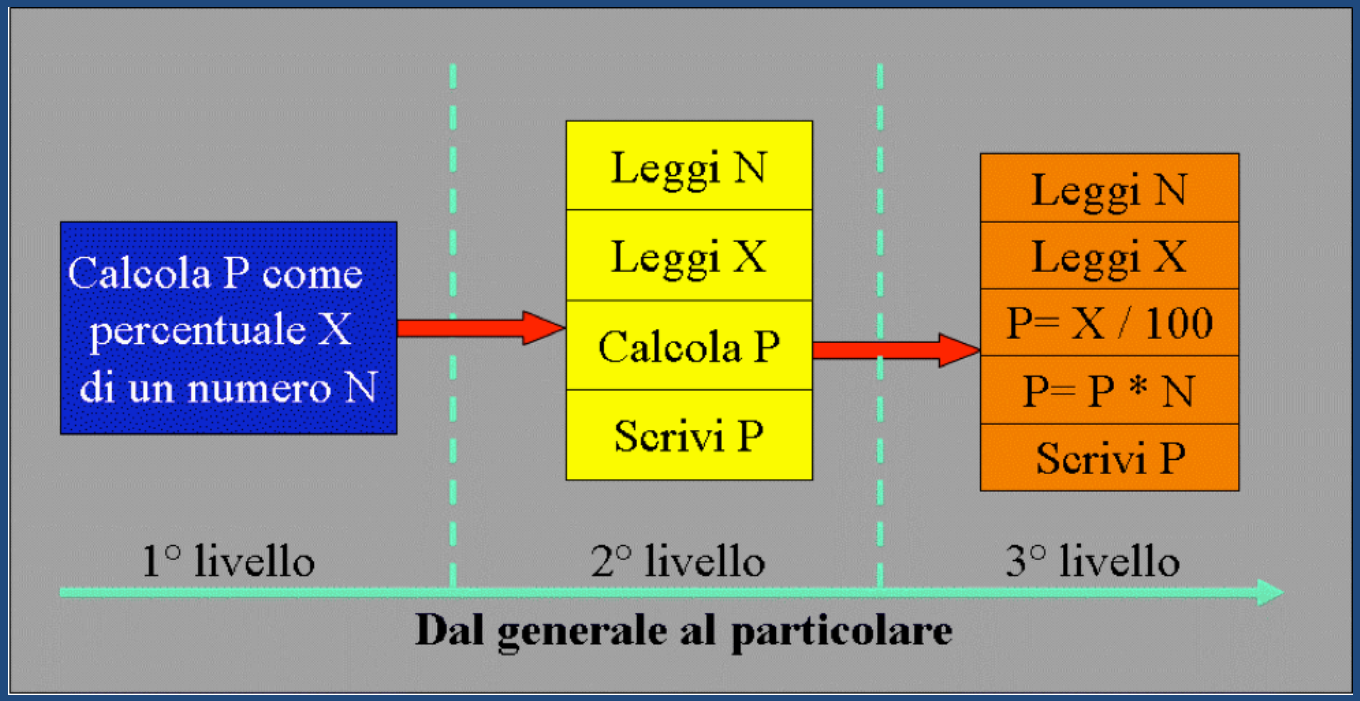
Caratteristiche: 1in 1out

- Le strutture di controllo sono da considerarsi degli schemi di composizione dei moduli costituenti il programma.
- Esse devono assolutamente avere un solo ingresso ed una sola uscita.
  - Tale condizione discende dalla necessità che un modulo, composto dall'integrazione di altri moduli tramite le strutture di un controllo, abbia un solo punto di ingresso ed un solo punto di uscita così come i singoli moduli componenti.

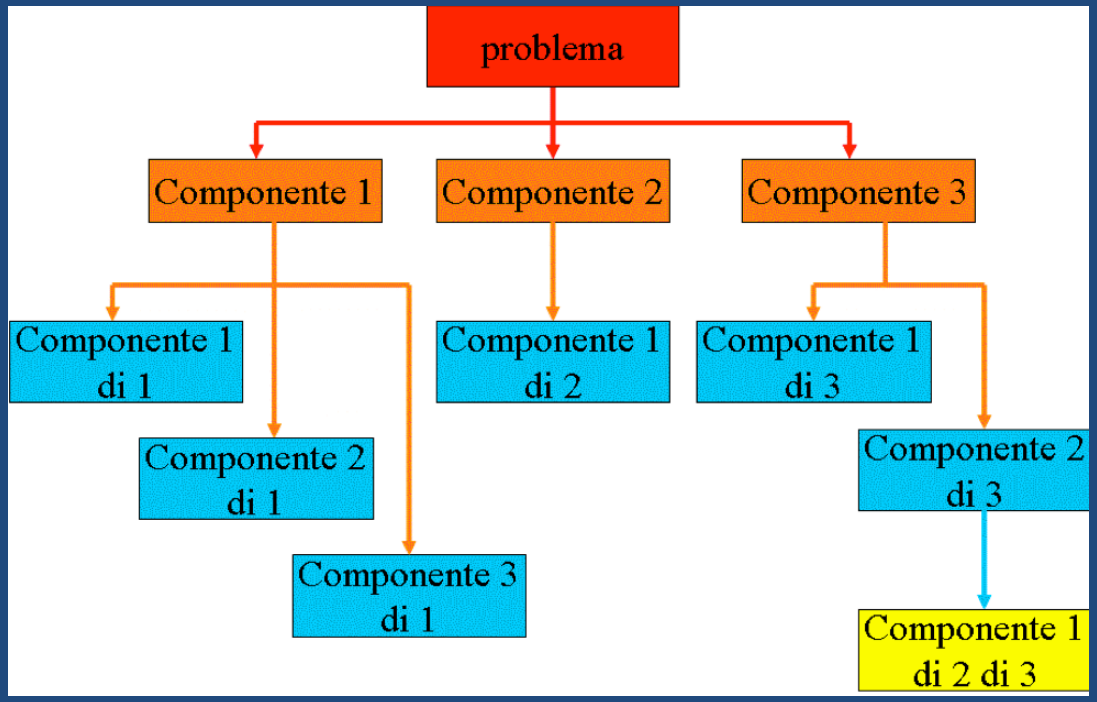
Caratteristiche: Top Down

- Il ***top-down*** e lo ***stepwise refinement*** costituiscono il modo procedurale di raggiungimento della soluzione (divide et impera)
  - Tale approccio parte dalla considerazione che la complessità di un problema da risolvere non consente di tener conto contemporaneamente di tutte le decisioni realizzative.
  - Sarà quindi necessario procedere per "Raffinamenti Successivi" procedendo dal generale al particolare.
- "*si analizza il problema al più alto livello possibile di astrazione individuandone gli elementi più importanti e supponendo di avere a disposizione un sistema adatto ad eseguire gli elementi funzionali individuati*".
  - Ogni elemento, a sua volta, diventa il problema da analizzare subendo una suddivisione in problemi più elementari.
  - Il procedimento continua fino a raggiungere un livello di scomposizione *comprensibile all'esecutore* (o software del sistema in uso).

Esempio top down



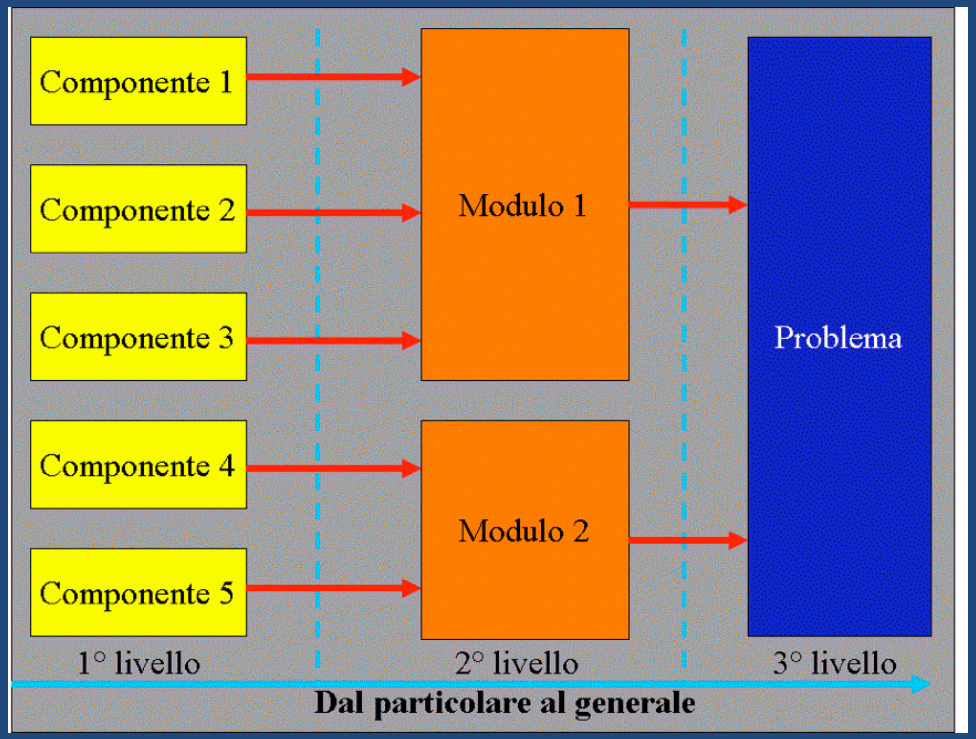
Soluzione del Problema: Albero



- la radice corrisponde al problema
- i nodi rappresentano le differenti decisioni di progetto,
- le foglie, vengono associate alla descrizione della soluzione in modo comprensibile all'esecutore.

Bottom Up

- … parte considerando il sistema reale a disposizione e creando man mano moduli elementari che opportunamente integrati formano moduli capaci di compiere operazioni più complesse.
- Il procedimento continua fino a quando è stato creato un modulo che corrisponde proprio alla soluzione del problema



La progettazione dei programmi di piccole dimensioni

- Una delle esigenze maggiormente sentita è quella di una separazione netta, in fase progettuale, tra il “cosa” (analisi dei requisiti e specifiche funzionali) e il “come” (progetto a diversi livelli di dettaglio).
  - In effetti la distinzione tra il “cosa” e il “come” è comune a qualsiasi tipo di progetto ed è un modo per esprimere con altre parole la separazione fra analisi e sintesi.

Fasi

- *fase di analisi dei requisiti e delle funzioni*
- *fase di progetto*
- *analisi critica della soluzione*

Analisi

- Capacità di acquisire tutte le informazioni necessarie alla comprensione del problema e di strutturarle in un modello che esprima, in un linguaggio adeguato, una rappresentazione coerente e completa di cosa si deve fare.
  - Le informazioni devono essere ricavate ..
    - dai colloqui con gli utenti di qualsiasi livello, ovvero con coloro che sono coinvolti nell'uso del programma sia a livello direttivo che esecutivo,
    - da un esame dell'ambiente in cui il programma sarà utilizzato,
- queste attività costituiscono la cosiddetta fase di definizione dei requisiti.
- Il compito dell'analista è quello di trarre, dall'insieme confuso e a volte contraddittorio di esigenze ed obiettivi, un **modello che esprima il problema** con un formalismo atto ad aumentarne la comprensibilità.

Requisiti

- i requisiti funzionali
  - cosa deve fare il programma e su quali dati deve operare
- i requisiti non funzionali
  - quali prestazioni il programma deve offrire
- Il passaggio dall'acquisizione dei requisiti alla formulazione del modello rappresenta la cosiddetta fase di analisi di un problema

Correttezza

- È necessario verificare che ogni aspetto delle specifiche abbia un'unica interpretazione nel contesto delle applicazioni che si descrivono ed effettuare delle esemplificazioni degli effetti delle trasformazioni tra dati di ingresso e dati di uscita.
  - Tali esemplificazioni costituiscono la base per un confronto del programma con l'utente che permette di verificare la correttezza delle specifiche.
- Sempre ai fini della correttezza occorrerà:
  - verificare che nelle specifiche non vi siano inconsistenze;
  - verificare che le specifiche coprano tutte le possibili situazioni coinvolte nel problema.
- Al termine della fase di analisi si deve disporre della documentazione su:
  - la definizione dei dati di ingresso al problema;
  - la definizione dei dati in uscita dal problema;
  - la descrizione di un metodo risolutivo che sulla carta risolva le specifiche del problema.

Progetto

- La fase di progetto può iniziare solo dopo un'accurata fase di definizione dei requisiti.
- Si articola nelle attività di *raffinamento successivo dei dati e dell'algoritmo*.
- Costruire un algoritmo equivale a:
  - esaminare una specifica realtà (il problema assegnato),
  - costruirne un'astrazione,
  - infine rappresentare una tale astrazione in maniera formale in un linguaggio di programmazione.
- La tecnica top-down si presenta come un approccio guida con l'obiettivo preciso di ridurre la complessità e di offrire uno strumento di composizione di un'architettura modulare dei programmi.
  - Il modo di procedere di tale approccio è una diretta applicazione del metodo analitico deduttivo che si è rilevato, storicamente, il più adatto alla media degli esseri umani

Come procedere

- Un modo di procedere è quello per livelli di astrazione.
- L'astrazione consiste nell'estrazione di dettagli essenziali mentre vengono omessi i dettagli non essenziali.
  - Ogni livello della decomposizione presenta una visione astratta dei livelli più bassi in cui i dettagli vengono spinti, quanto più possibile, ai livelli ancora più bassi.
  - I moduli di livello più alto specificano gli obiettivi di qualche azione oppure cosa deve essere fatto, mentre quelli di livello più basso descrivono come l'obiettivo verrà raggiunto.
- Il passo di raffinamento iterativo produce, a partire dal problema, una prima decomposizione dell'algoritmo e prosegue con successivi raffinamenti, sempre più vicini all'espressione finale dell'algoritmo nel linguaggio di programmazione.

Separazione Analisi e Progetto

- Motivi alla base di tale separazione sono:
  - la possibilità di documentare in maniera completa i requisiti del problema e quindi, indirettamente, le scelte della fase di progetto;
  - l'impossibilità d'inficiare i risultati dell'analisi a causa di scelte anticipate di progetto;
  - la possibilità di rendere indipendente l'analisi dai vincoli di realizzazione.

Dati e Funzioni

- Un'altra distinzione, in senso orizzontale, avviene fra dati e funzioni che rappresentano gli aspetti duali, ma strettamente legati, di ogni problema che si esamina;
- tale separazione è dovuta
  - sia a ragioni intrinseche
    - in quanto dati e funzioni sono caratterizzati da proprietà differenti,
  - che a ragioni metodologiche
    - in quanto è opportuno trattare separatamente funzioni da un lato e dati dall'altro, in modo da ottimizzarne gli aspetti relativi in maniera indipendente dai condizionamenti che, viceversa, un'analisi complessiva comporterebbe.
- Ciò consente di ottenere:
  - l'esame accurato dell'aspetto semantico dei dati;
  - la completezza dell'analisi, in quanto essa viene compiuta da differenti punti di vista;
  - l'assicurazione che dati e funzioni assumano il giusto peso in analisi e non si enfatizzino gli uni rispetto agli altri.

La Documentazione dei Programmi

- La documentazione dei programmi è lo strumento fondamentale per la *chiarezza* e *la leggibilità* dei programmi.
- Tali caratteristiche di leggibilità consentono:
  - una più semplice comprensione di quale sia il problema che il programma risolve e quindi della correttezza del programma stesso;
  - una più semplice prosecuzione del progetto ogni qualvolta lo si sia interrotto;
  - una più elementare comunicazione delle scelte di progetto;
  - una più semplice modificabilità del programma al variare delle specifiche del problema.

Regole della documentazione

- Produrre la documentazione nel corso stesso del progetto
  - in quanto si può essere certi della efficacia e della completezza della documentazione soltanto se si documentano le scelte nel momento in cui esse vengono fatte.
- Inserire la documentazione quanto più possibile all'interno del programma in modo che viva con il programma stesso, cambi e cresca con esso, e quindi sia sempre aggiornata.

Documentazione esterna

- E’ il primo livello di documentazione e va compilato preliminarmente nella fase di analisi dei requisiti.
- Descrive soltanto il "cosa" fa il programma e non il "come" lo fa.
- Deve segnalare dettagli operativi tali da rendere autonomo l'utente del programma nell'uso dello stesso.
  - a) funzionalità esterne;
  - b) attivazione del programma;
  - c) diagnostiche di errore;
  - d) configurazione richiesta del sistema;
  - e) indicazione sull'installazione del programma;
  - f) versione e data di aggiornamento.

Documentazione interna

- descrive la struttura interna del programma in termini di scelte sulle strutture dati e sull'algoritmo.
  - evidenziazione della struttura del programma mediante l'uso dell'indentazione
    - indentare un programma significa mettere in evidenza nel testo del programma, mediante opportuni incolonnamenti, blocchi di istruzioni che svolgono una funzionalità ben precisa
  - documentazione top-down
    - ossia facendo comprendere come il programma è stato generato attraverso i vari raffinamenti
  - uso di nomi di variabili autoesplicativi, ossia tali da spiegare il ruolo da esse svolte nel corpo del programma
  - commento del programma attraverso le frasi di commento di cui tutti i linguaggi di programmazione sono dotati