Software, Algoritmi e Programmi

Lezione n. 3

Obiettivi

- 3.1 Definizione di Software
- 3.2 Definizione di algoritmo, sue proprietà e formalismi
- 3.3 Metodo top-down, e programmazione strutturata
- 3.4 Definizione di programma e il processo di programmazione
- 3.5 Complessità degli algoritmi

Software

parte n. 3.1

Metodo del *problem solving* e sviluppo del software

- Il *problem solving* è l'insieme di metodi formali per definire e risolvere un problema
- E' un metodo usata dall'uomo per risolvere tutti i tipi di problemi (economici, statistici, giuridici, etc.)
- Tecnica usata anche per sviluppare *software*
- Esempio di *problem solving*
  - Problema - Prelevare contanti in banca
  - Analisi - si possono prelevare contanti in diversi modi:bancomat, mediante assegno, chiedendo un prestito, fare una rapina, etc.
  - Soluzioni - si decide per il bancomat, si descrivono i passi operativi e le istruzioni che attivano il bancomat
  - Elaborazione - esecuzione delle operazioni
  - Risultati - i contanti

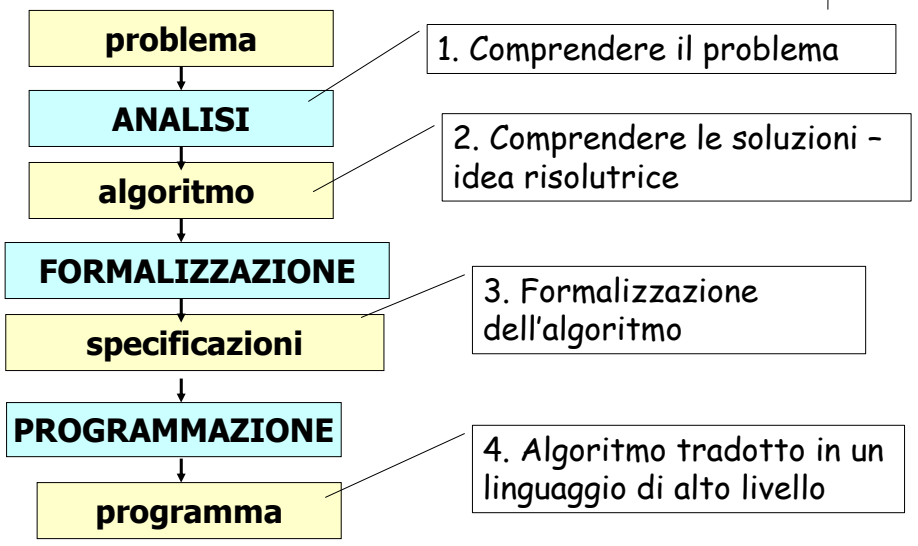
Sviluppare software

- Sviluppare software per un calcolatore, ossia per un esecutore-automa, e' un'attivita' di "**risoluzione di problemi**" secondo il metodo del problem solving
- Essa è divisa in fasi:

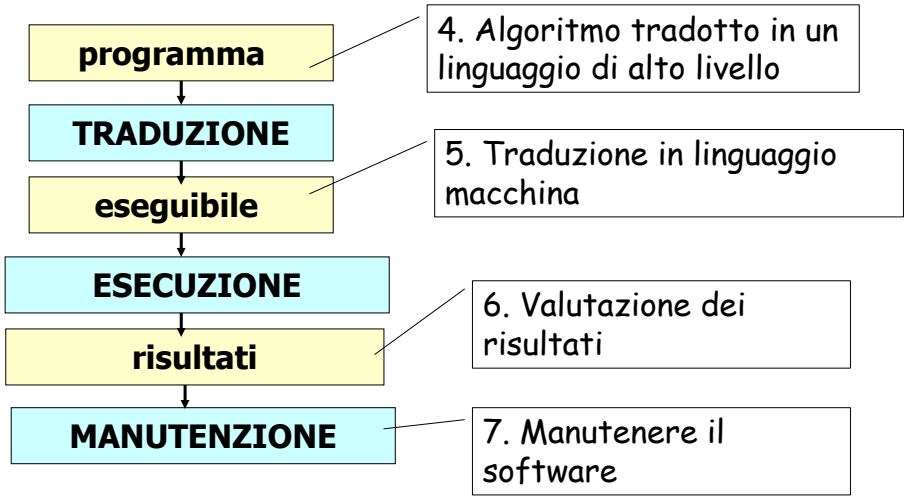
Fasi di risoluzione di un problema con il calcolatore:

1. Analizzare il problema da risolvere
2. Avere l'idea risolutiva
3. **Scrivere l'algoritmo formalizzandolo**
4. **Implementare l'algoritmo in un programma di alto livello**
5. **Tradurlo in linugaggio macchina**
6. Verificare la correttezza del programma
7. Documentare, mantenere e aggiornare il programma

Creazione di un software: dal problema ai risultati (1/2)



Creazione di un software: dal problema ai risultati (2/2)

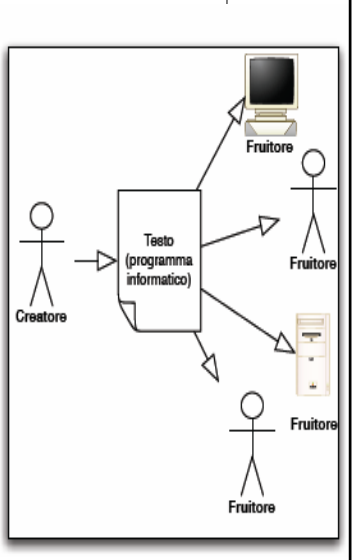


Un esecutore particolare: il calcolatore

- Se per un esecutore umano le fasi di formalizzazione, programmazione, traduzione in azioni (fasi n.3,4,5) avvengono con una elaborazione cognitiva
- Per il calcolatore occorrono tre fasi intermedie poiché non è dotato di cognizione autonoma:
  - Formalizzazione dell'algoritmo secondo un metodo non ambiguo
  - Programmazione ad alto livello comprensibile al calcolatore
  - Traduzione in linguaggio comprensibile al calcolatore ossia in linguaggio macchina

Programma & Software, Testi & Ordini

- Anche la fase di l'esecuzione del programma necessita di alcune riflessioni
- Prima del calcolatore il testo era scritto dall'uomo per l'uomo
- Le operazioni eseguite dall'uomo o da macchine prive di capacità computazionale universale
- Con l'avvento del calcolatore:
  - i programmi sono testi con una particolare proprietà ossia capaci di impartire ordini al calcolatore
  - testi creati dall'uomo che rendono il calcolatore capace di elaborazioni autonome, astratte, generali rispetto ad una classe di problemi



Software – definizione informatica

- "Istruzioni che eseguite da un computer svolgono una funzione prestabilita con prestazioni prestabilite - (programma di alto livello ed eseguibile)
- strutture dati mediante le quali i programmi trattano adeguatamente le informazioni - (schemi logici e fisici dei dati)
- documenti che descrivono le operazioni e l'uso dei programmi - (documentazione tecnica e manuale utente)"

(R.S. Pressman,Principi di Ingegneria del software, McGraw-Hill 2000)

Software: definizione per livelli

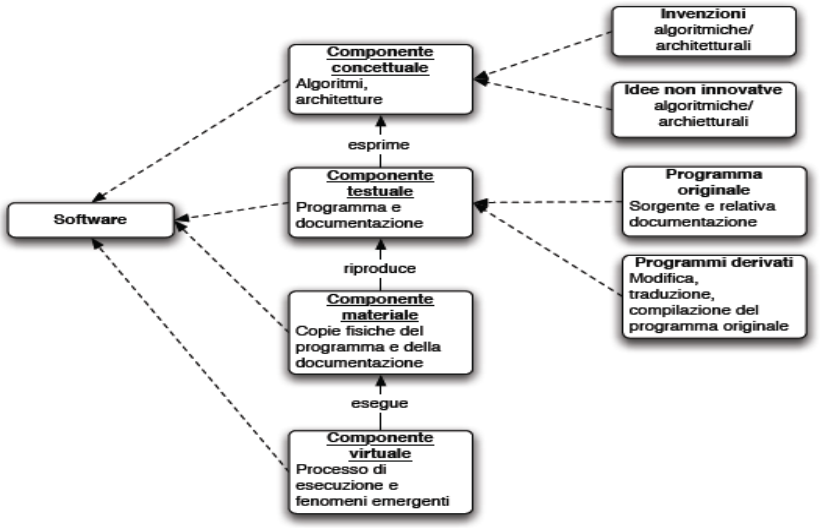
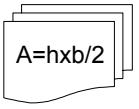
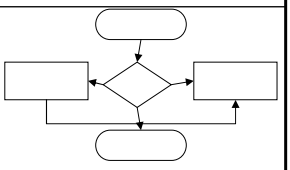
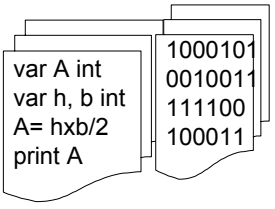


Figura 3.14: *Gli aspetti del software*

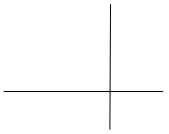
Il Software usando il Framework di Zachman

Livello concettuale	Algoritmo	
Livello logico	Formalizzazione/ documentazione	
Livello fisico	Linguaggio alto livello, linguaggio macchina, esecuzione	

Software: la definizione

- Software: insieme di programmi scritti in qualche linguaggio di programmazione *eseguibili* dal computer (tutelato con il diritto d'autore anche se..)
  - software ≠ algoritmo
  - software ≠ programma
  - algoritmo ≠ programma
  - software = algoritmo + programmi & documentazione + file fisici eseguibili + l'azione di esecuzione

Hardware e firmware: definizioni



- Hardware: parte fisica del computer costituita da parti elettroniche e meccaniche (tutelato con il brevetto)
- Firmware: insieme di microprogrammi registrati sulle memorie permanenti dei dispositivi elettronici, solitamente introdotti dal costruttore e cablati nell'hardware (tutelato con il brevetto)

Strati del software

Tipologia del software

- Software di base
  - software al servizio di altri software
  - esempio il sistema operativo:
    - insieme di programmi che governano le funzioni e le risorse primarie del calcolatore
    - esecuzione di programmi
    - operazioni di ingresso/uscita
    - gestione di file
    - protezione
    - rilevazione errori
- Software applicativo: dedicato ad uno scopo applicativo

Tipologie di software

- software real-time - software dedicato alla sorveglianza, all'analisi e all'elaborazione di eventi esterni (rilevamento di temperature di una piastra di acciaio durante la lavorazione, pilota automatico, sala operatoria)
- software gestionale - elaborazione dei dati e dei processi aziendali - 70%-80%
- software scientifico - astronomia, calcolo parallelo, etc.
- software di Intelligenza Artificiale - sistemi esperti, reti neurali, dimostratori di teoremi, alcuni sono dotati di autonomia, reattività e pro-attività (es. agenti intelligenti)
- software *embedded* - programmi residenti in prodotti industriali (lavatrici, forno, termostati ambientali, etc.)
- software per PC - applicativi di *office-automation* (fogli elettronici, elaboratori di testi, etc.)
- software basato su Internet - B2B, B2C, portali, etc. spesso erogato mediante servizi e non prodotti

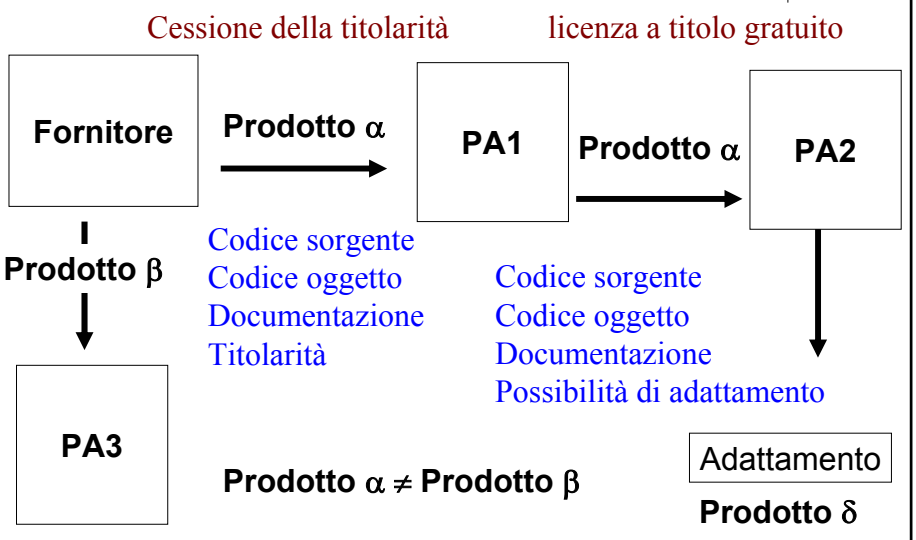
Categorie di software

- Software Generici
  - prodotti software standardizzati venduti sul mercato
  - le specifiche vengono dettate dal mercato e dal produttore stesso
- Software Dedicati o *ad hoc*
  - progetti ad hoc creati per un determinato cliente
  - le specifiche vengono dettate dal cliente
- Modello di business diverso
  - Prodotto (generici) vs. Progetto (ad hoc)
  - Licenza (generici) vs. Contratto ad oggetto informatico (ad hoc)
  - Metafora del prodotto industriale vs. quello artigianale

Obblighi per la PA in materia di sviluppo software

- Per la PA vi sono norme che OBBLIGANO le amministrazioni a richiedere ed ottenere la titolarità dei software “ad hoc”
- ex Art. 5, Direttiva del 19 dicembre 2003 “Sviluppo ed utilizzazione dei programmi informatici da parte delle pubbliche amministrazioni” (Gazzetta Ufficiale n. 31 del 7-2-2004)
- e a rilasciare in “uso gratuito” tale software alle altre PA (ex Art. 69, D.lgs. 82/2005 – CAD – Codice dell’Amministrazione Digitale)

Possibile scenario



Il riuso e la crisi del modello di business (1/2)

- Il riuso del software nella PA è stato introdotto per:
  - razionalizzare la spesa pubblica in tema di servizi informatici
  - incentivare il riuso piuttosto che duplicare gli acquisti
  - rendere autonome le PA di poter modificare, integrare, aggiornare i software senza un legame vincolante con il fornitore

Il riuso e la crisi del modello di business (2/2)

- Il riuso accompagnato al fenomeno dell'open source ha fortemente modificato l'industria del software che da un modello basato principalmente sulla licenza d'uso a pagamento (es. Microsoft) è passata a due nuovi modelli di business:
  - open source – basato principalmente sulla vendita di personalizzazioni e di giornate uomo di assistenza
  - servizi ASP – *application service provided* vendita di servizi via rete
    (es. GoogleDocs, hosting di software, etc.)
- Sulla base di questi due nuovi modelli di business sono nati nuovi strumenti giuridici di tutela del software
  - GPL e LGPL – licenze dell'open source
  - Contratti/Appalti di servizi ASP

Materiali di riferimento e Domande possibili

- Capitolo 3 del Sartor
- Definizione di software, hardware e firmware
- Tipi di software e tipi di modelli di business
- Relazione fra software, algoritmo e programma
- Il problem solving e il software
- Le fasi di produzione di un software
- Il riuso e il software ad hoc nella normativa italiana (CAD e circolare Stanca)
- Nuovi modelli di business: open source a ASP

Algoritmo

parte n. 3.2

Informatica – definizione dell'ACM

- Come si è visto l'algoritmo è una delle tante fasi della creazione del software e rappresenta la *fase concettuale*
- La fase dedicata alla formalizzazione delle soluzioni in informatica si realizza nella creazione di algoritmi
- Del resto la definizione data dall'ACM dell'informatica è:

  "L'Informatica è lo studio sistematico degli algoritmi che descrivono e trasformano l'informazione: la loro teoria, analisi, progetto, efficienza, realizzazione e applicazione."
  (ACM - Association for Computing Machinery)

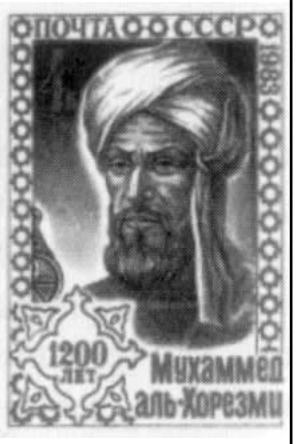
Algoritmo - definizione intuitiva

Elenco preciso di operazioni, comprensibile da un esecutore, che definisce una sequenza finita di passi i quali risolvono ogni problema di un dato tipo (classe di problemi).

Esempio: operazioni necessarie per compiere una telefonata, per prelevare denaro dal bancomat, per iscriversi ad un esame, etc.

Origini dell'algoritmo

- Il concetto di algoritmo è antico e non è strettamente legato al calcolatore: l'esecutore può essere diverso
- Sono stati ritrovati algoritmi in tavolette antiche in Mesopotamia risalenti al 1800-1600 a.c.
- Il termine algoritmo è la latinizzazione dal nome di un matematico persiano – Al-Khuwarizmi – vissuto nel nono secolo d.c. (*Algoritmi de numero Indorum* – versione latina, trattato sull'algebra dei numeri arabo-indiani)

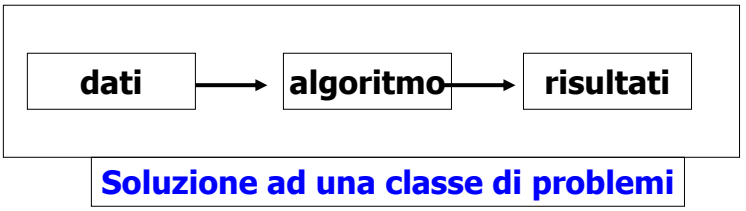


Esempio – prelevo contanti dal bancomat, macro operazioni

1. Inserimento della tessera nell'apposito macchinario
2. Inserimento del codice segreto
3. Scegli importo
4 Scelta operazione
   4.1 Se l'operazione è possibile allora esegui l'operazione
   Altrimenti
   4.2 Visualizza messaggio di errore
5. Conclusione
6. Prelievo tessera
7. Prelievo contanti

Algoritmo - definizione rigorosa

Sequenza ordinata finita di passi, ripetibili e non ambigui, che se eseguita con determinati dati in ingresso (input) produce in uscita(output) dei risultati ovvero la soluzione di una classe di problemi



Proprietà di un algoritmo

- Finitezza - deve portare alla soluzione in un numero finito di passi
- Generalità - per classi di problemi
- Ripetitività - con gli stessi dati deve fornire gli stessi risultati
- Determinismo o Non ambiguità – non dipende dall'esecutore, ossia le azioni sono non ambigue e se eseguite con gli stessi dati da persone diverse si ottengono sempre gli stessi risultati

Una ricetta non è un algoritmo

- di solito fra gli ingredienti vi sono espressioni ambigue come “un pizzico di sale” o “quanto basta” quindi lasciati alla soggettività
- la descrizione delle azioni non sono rigorose e necessitano interpretazioni “temperare il coccolato” o “fare un impasto base per torte”
  – violazione del principio di determinismo
- non è vero che ripetendo gli stessi passi si ottengono gli stessi risultati
  – violazione del principio di ripetitività

Dati soggettivi +istruzioni+esecutore = risultati

- Un algoritmo si suppone sempre che comunichi con l’ambiente acquisendo dati (dati soggettivi) e restituendo risultati
- L’algoritmo è composto da istruzioni che operano su dati prodotti dall’algoritmo stesso o acquisiti dall’esterno
- L’algoritmo deve essere eseguito da un esecutore (calcolatrice, uomo, meccanismo, ingranaggio meccanico, etc. non necessariamente dal computer)
- Occorre quindi descrivere le istruzioni utilizzando un linguaggio preciso e generale comprensibile all’esecutore

dati→ istruzione→esecutore→risultati

Istruzioni = azioni+dati oggettivi

- Le istruzioni sono composte da due parti:
  - azione - descrizione delle operazioni
  - dati - descrizione degli oggetti manipolati dalle operazioni (dati oggettivi)
- esempio
  - Inserisci [azione] la tessera [dato]
  - digitare [azione] codice segreto [dato]
  - selezionare [azione] importo [dato]
- vi sono istruzioni zerarie, unarie, binarie, ternarie
  - start
  - inizializza A
  - metti A in B
  - somma A e B in C

Rappresentazione logica degli algoritmi: due tecniche

- pseudocodifica (o pseudocodice)
  - professionale
    - verbi di ”esecuzione” (effetto osservabile)
    - condizioni
    - iterazioni
- diagramma a blocchi
  - utile a scopi dimostrativi
    - indica un flusso di istruzioni ovvero la sequenza dei passi da eseguire
    - basato su simboli grafici
    - ogni simbolo corrisponde a un costrutto
- le due modalità sono semanticamente equivalenti

Pseudocodifica

- Descrive l'algoritmo con il linguaggio naturale semplificato al fine di togliere le ambiguità
- La descrizione mediante la pseudocodifica si suddivide in due parti:
  - dichiarazione delle variabili
  - dichiarazione delle azioni

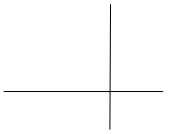
Pseudocodifica - esempio

```
Inizio
Inizializza MAX a 0
Inizializza N1 a 0
Leggi N1
Finché N1 != 999
    Se N1 > MAX allora
            Assegna N1 a MAX
    Leggi N1
Stampa MAX
Fine
```

I diagrammi a blocchi – (1/6)

- la diagrammazione a blocchi o flow chart è un metodo per rappresentare l'algoritmo in modo grafico sintetico e preciso
- un diagramma a blocchi indica un flusso di istruzioni ovvero la sequenza dei passi da eseguire
- è basato su simboli grafici
- ogni simbolo corrisponde ad un preciso costrutto o insieme di istruzioni

I diagrammi a blocchi – (2/6)

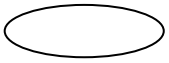


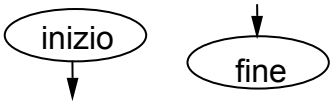
- Un diagramma a blocchi è un insieme di blocchi elementari costituito sempre dalle seguenti parti:
  - blocco di inizio
  - blocco di fine
  - numero finito di blocchi di lettura/scrittura o di blocchi operativi
  - numero finito di blocchi di controllo (opzionale)

I diagrammi a blocchi – (3/6)

Rappresenta il flusso (l'ordine) del diagramma

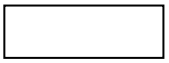
Istruzioni di inizio e di fine





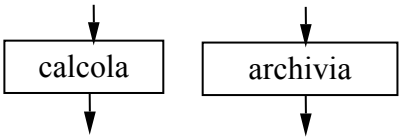
I diagrammi a blocchi – (4/6)

istruzione operativa (*effettiva*)



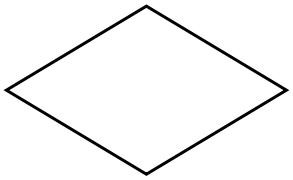
Rappresenta una elaborazione

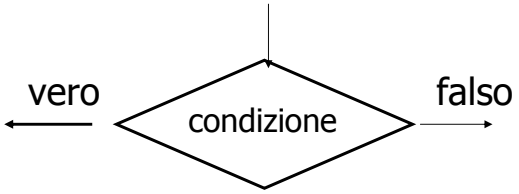
Esempi:



I diagrammi a blocchi – (5/6)

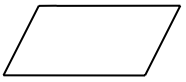
istruzioni di controllo



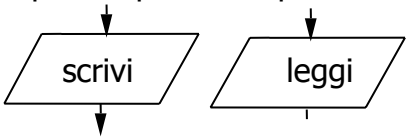


I diagrammi a blocchi – (6/6)

istruzione di input/output



Rappresenta un'operazione di input/output. Esempi:

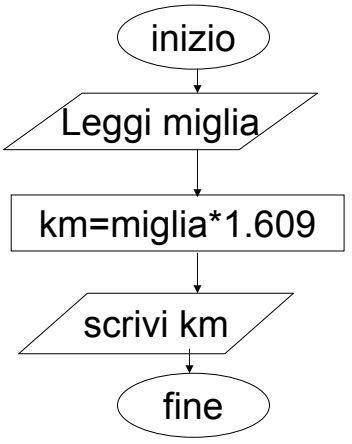


Esempio: conversione miglia-chilometri

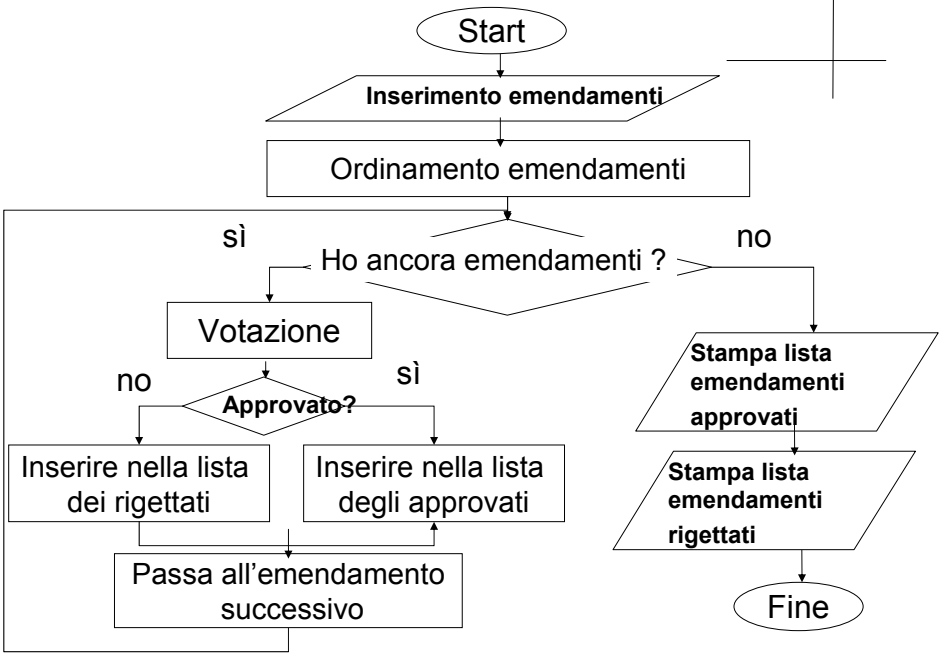
Algoritmo

1.Inizio
2.Leggi miglia
3. Km=miglia*1.609
4. Scrivi km
5. Fine

Diagramma a blocchi



Es. Votazione emendamenti alla Camera



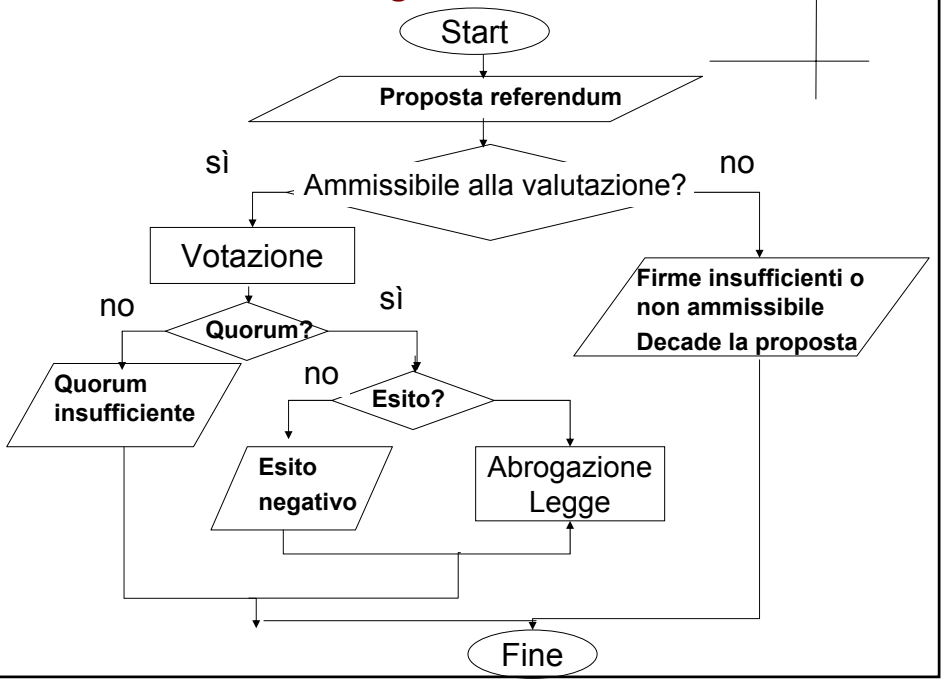
Spiegazione esempio

- Vi sono istruzioni di input/ouput: inserimento emendamenti e stampa liste
- Vi sono istruzioni operative: ordinamento, votazione, inserimento nella lista, etc.
- Vi è un blocco di istruzioni ripetuto finché non si esauriscono gli emendamenti (ciclo)
- Vi è un confronto (if): emendamento approvato o no
- Vi è una solo inizio e una sola fine

Art. 75 Cost. Referendum abrogativo

- 1. Proposta a cura di 5 regioni o da parte di 500.000 cittadini
- 2. Proposta ammissibile valutata della Corte Costituzionale
  - 2.1 allora si fissa la data di votazione
  - 2.2 altrimenti decade la proposta
- 3. Se si vota
  - 3.1 si raggiunge il quorum
    - 3.1.1 vincono i sì, allora si abroga la legge
    - 3.1.2 vincono i no, allora non si abroga la legge
  - 3.2 non si è raggiunto il quorum
- 4. Fine

Es. Referendum abrogativo



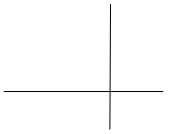
Tipi di istruzioni

parte n. 3.3

Tipi di istruzioni

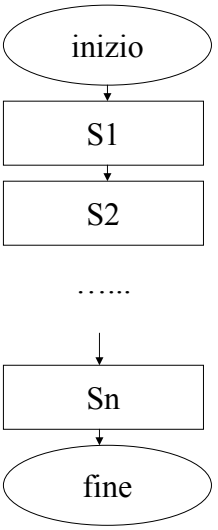
- La sequenza
- Istruzione di selezione
- Istruzioni di ciclo
- Istruzioni di salto

La sequenza



- Le istruzioni si susseguono rispetto ad un ordine e vengono eseguite nella sequenza indicata
- {<S1>; <S2>; … <Sn>}

La sequenza



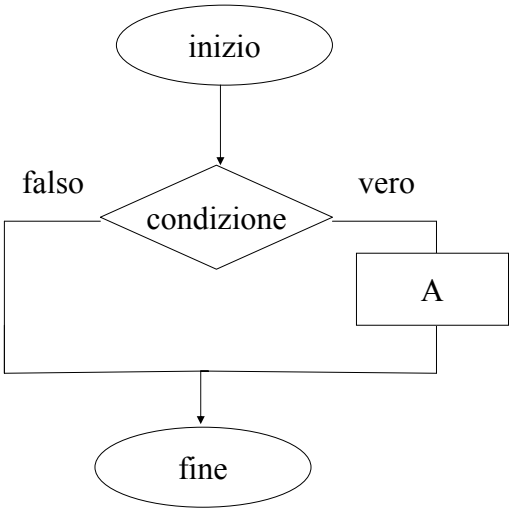
La selezione

- If (<condizione>)
  {
  <S1>…..
  <Sn>
  }

  BLOCCO

  - Se la “condizione” è vera allora vengono eseguite l’istruzioni del blocco
  - Se la “condizione” e’ falsa, allora il blocco non viene eseguito

La selezione

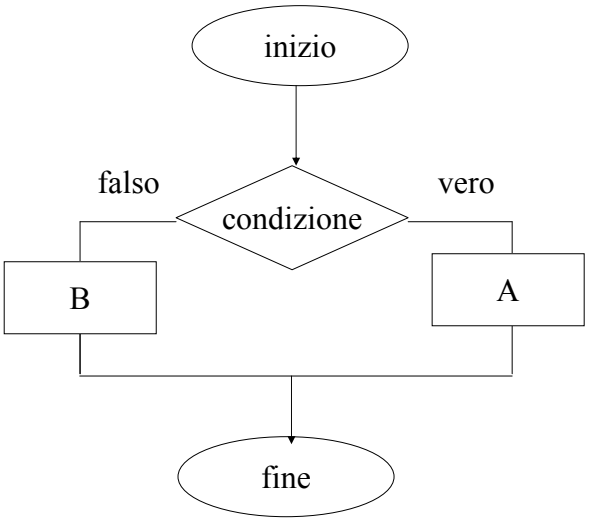


La selezione con alternativa

- If (<condizione>)
  then
  {
  <blocco1>
  }
  else
  {
  <blocco2>
  }

  - Se la condizione è vera allora viene eseguito il blocco1
  - altrimenti il blocco2

La selezione con alternativa

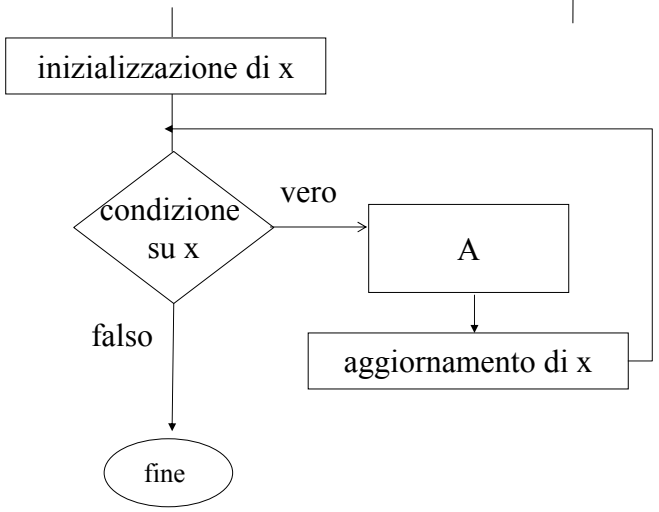


La ripetizione (while)

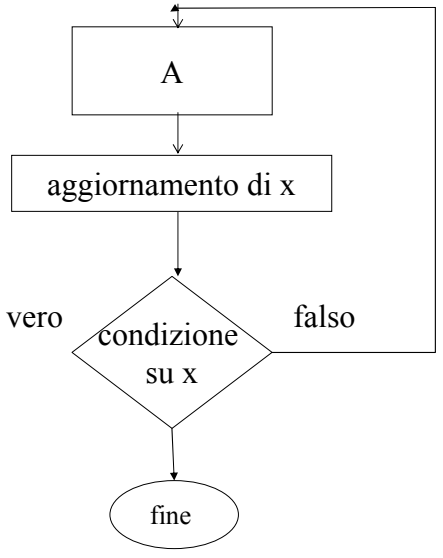
```
While (<condizione>)
  do
  {
  <blocco>
  }
```

- il blocco viene ripetuto finché la condizione risulta vera
- quando l'espressione risulta falsa la ripetizione cessa e si passa all'istruzione successiva

La ripetizione con la condizione vera in testa



La ripetizione con la condizione falsa in coda

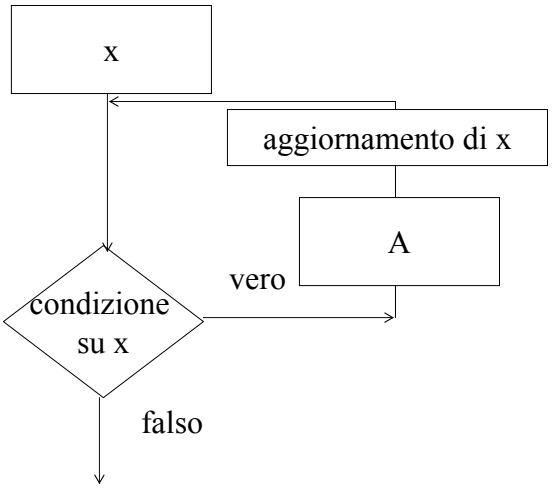


La ripetizione enumerativa (for)

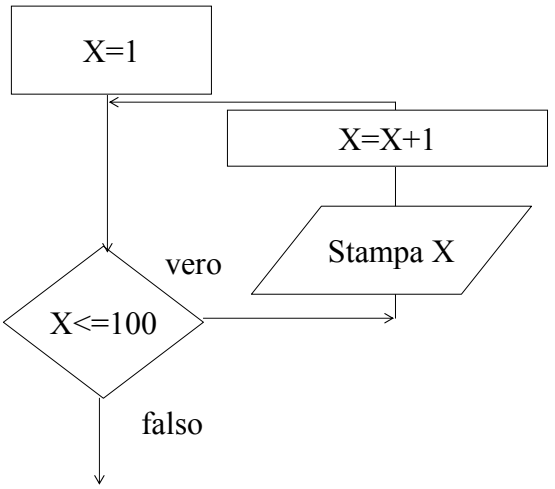
for (<espressione iniziale>, <condizione>, <aggiornameto espressione>)
{
<blocco>
}

- il blocco viene ripetuto finché la condizione risulta vera
- la formula di aggiornamento modifica i valori di confronto nell'espressione fino a quando la condizione non diviene vera

La ripetizione enumerativa (for)



Esempio di for



Cosa produce questo algoritmo ?

Istruzioni di controllo: i salti

- Istruzione di salto (*jump, goto*):
  - prescrive che l'ulteriore elaborazione continui a partire da un certo punto
  - trasferisce il controllo al punto indicato

Salti incondizionati e condizionati

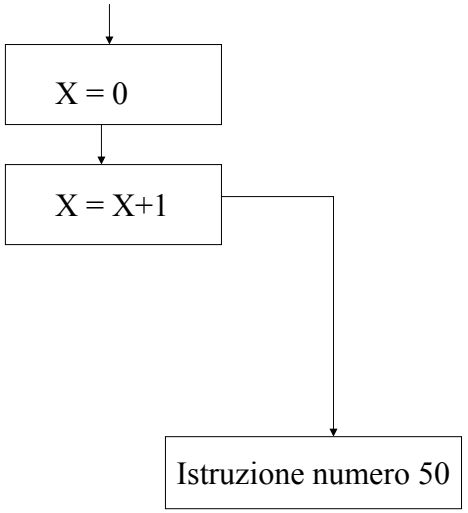
Salto incondizionato

1. X = 0;
2. X = X+1;
3. Goto 50

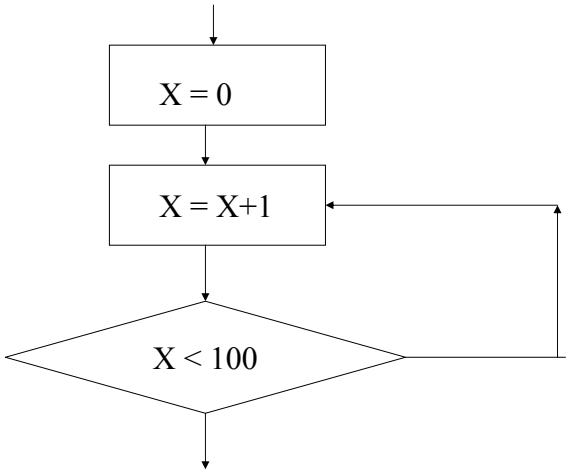
Soluzione: Salto condizionato

1. X = 0;
2. X = X+1;
3. If X < 10 go to 2

Salto incondizionato



Salto condizionato



Esercizi con i diagrammi a blocchi

- Modellare il processo di codecisione del Parlamento Europeo e del Consiglio dell'Unione Europea
- Modellare il procedimento di votazione di una legge costituzionale
- Modellare il procedimento di iscrizione a ruolo di una causa civile
- Modellare un procedimento amministrativo
- Modellare il procedimento legislativo anglosassone

Sviluppo strutturato degli algoritmi e metodologia top-down

Lezione n. 3.4

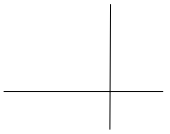
Tecniche di programmazione: verso la qualità del software

- Negli anni ’70 la programmazione avveniva come un’operazione artigianale e i metodi erano non omogenei, frammentari, con molti salti incondizionati (goto)
- Negli anni ’80 si ipotizzano delle tecniche per migliorare la qualità del software:
  - Metodo Top-down
  - Programmazione strutturata

Scomposizione in sotto-algoritmi: metodo top-down

- Spesso per semplificare il flusso procedurale si rappresentano insieme più istruzioni in un solo blocco grezzo concettualmente omogeneo
- Successivamente si scompone il blocco in istruzioni sempre più “fini”
- Si procede dal generale al particolare
- Questo metodo di analizzare i problemi partendo da macro-blocchi per poi arrivare alle istruzioni più dettagliate è detto metodo top-down

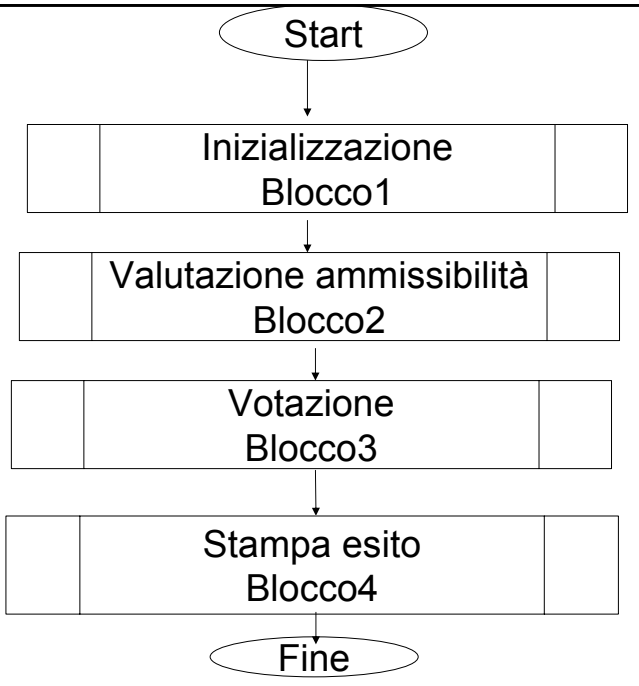
Il metodo top-down



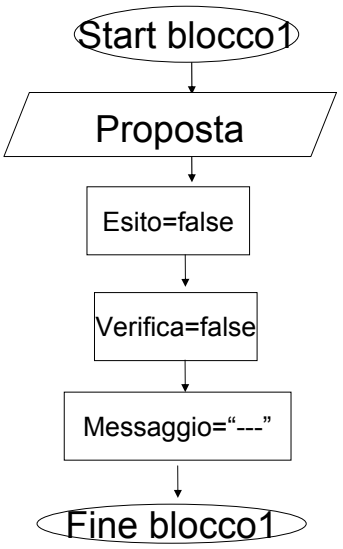
- Metodo top-down: scomposizione progressiva del problema e delle azioni (istruzioni) che lo risolvono
- Favorisce la divisione del lavoro:
  - Analisi: definizione della struttura generale del programma, dei moduli principali che lo compongono, e delle operazioni astratte svolte da ciascuno di essi, le cosiddette “specifiche”
  - Programmazione: realizzazione dei programmi che svolgono le singole operazioni individuate nella fase di analisi

Art. 75 Cost. Referendum abrogativo

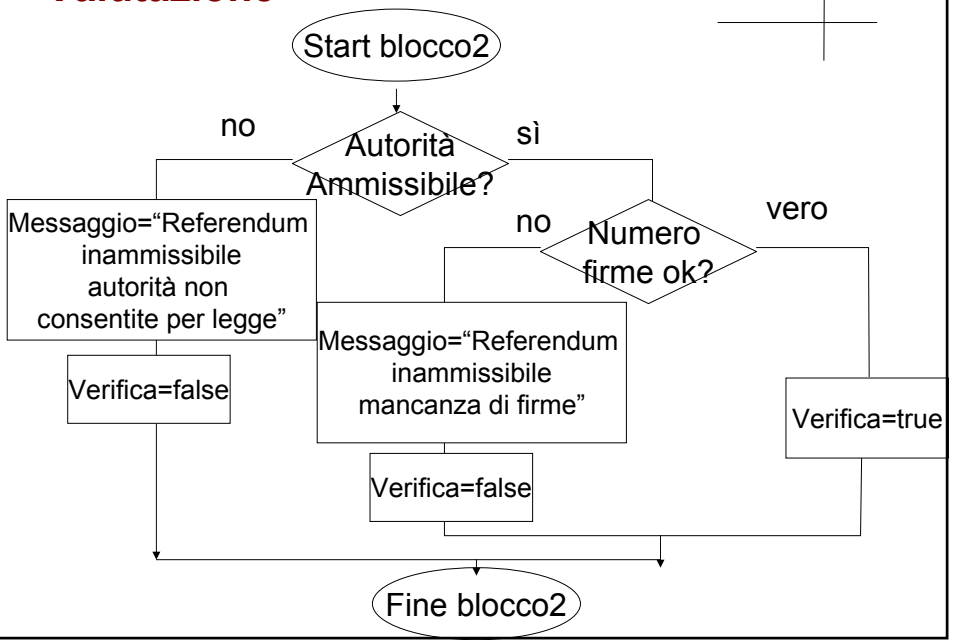
- 1. Proposta
- 2. Valutazione di ammissibilità della Corte Costituzionale
- 3. Votazione
- 4. Controllo dell'esito
- 5. Fine



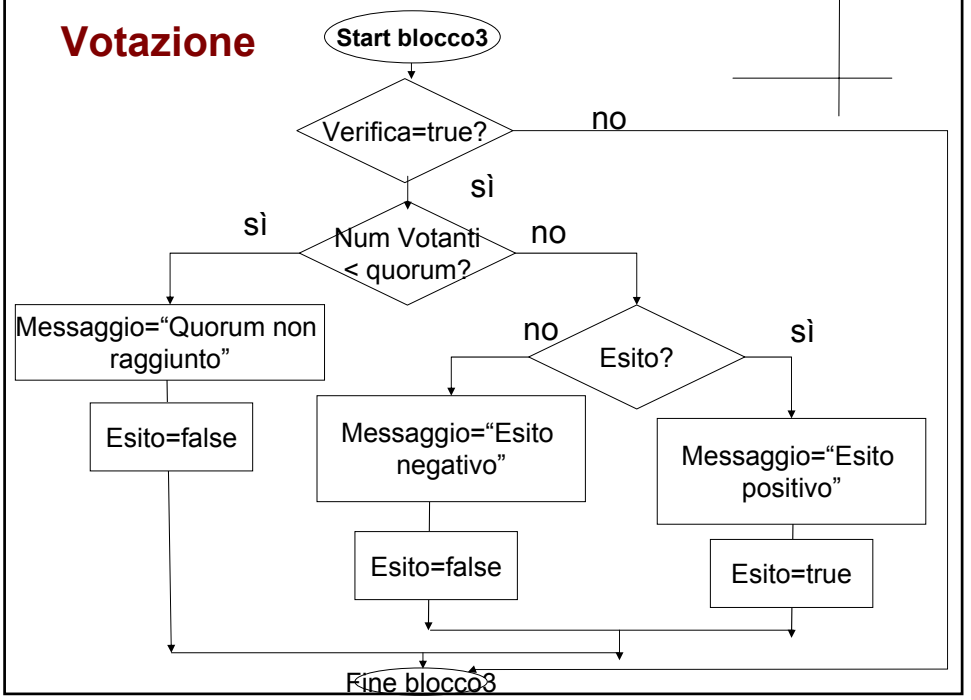
Inizializzazione



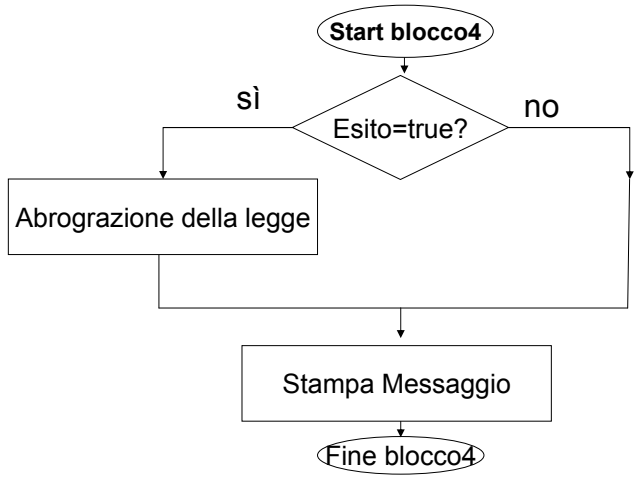
Valutazione



Votazione



Esito



Programmazione strutturata

- la programmazione strutturata è quel procedimento che permette di ottenere algoritmi facilmente documentabili e comprensibili, manutenibili e di buona qualità
- Si utilizzano solo tre tipi di istruzioni (costrutti):
  - Sequenza
  - Selezione
  - Ripetizione : tre tipi di ciclo condizione in testa (while), condizione in coda (repeat), ripetizione enumerativa (for)
- Regole base:
  - i salti (goto) sono rigorosamente proibiti
  - esiste un solo inizio e una sola fine di tutto il programma
  - le selezioni si chiudono sempre

Potenza della programmazione strutturata

- Le tre strutture presentate consentono di esprimere qualsiasi algoritmo
- Teorema di Bohm-Jacopini:

"Ogni diagramma a blocchi non strutturato è sempre trasformabile in un diagramma a blocchi strutturato equivalente..."

(...con l'eventuale aggiunta di una variabile)

Materiali di riferimento e Domande possibili

- Capitolo 3 del Sartor
- Definizione di algoritmo e sue caratteristiche
- Formalizzazione di un algoritmo secondo la pseudocodifica e i diagrammi a blocchi
- Cosa è la sequenza, una condizione, un ciclo, un salto
- I principali tipi di istruzioni secondo la grafica dei diagrammi a blocchi
- Cosa è una inizializzazione
- La programmazione strutturata: caratteristiche e obiettivi
- Il metodo top-down: caratteristiche e obiettivi

Complessità computazionale degli algoritmi

Lezione n. 3.5

I precursori dei calcolatore

- Calcolatore di Rodi o di Andikithira 65 a.C.
- Blaise Pascale – pascalina XVII secolo
- Gottfried Leibniz
- Joseh Jacquard XVIII secolo
- Charles Babbge XIX secolo
- Alan Turing XX secolo - Colossus
- John Von Neumann – macchina programmabile universale - ENIAC

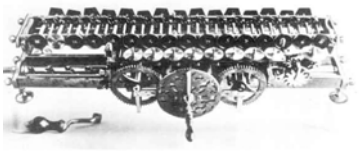


Figura 2.1: *La macchina calcolatrice di Leibniz*

Macchina di Babbage

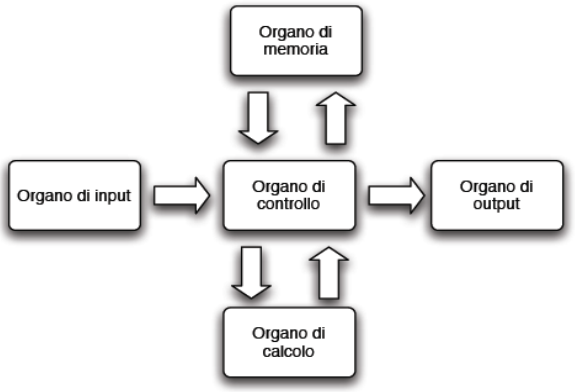


Figura 2.2: *Componenti del motore analitico di Babbage (e del moderno elaboratore)*

Il programma era già cablato all'interno del calcolatore

La macchina di von Neumann (1)

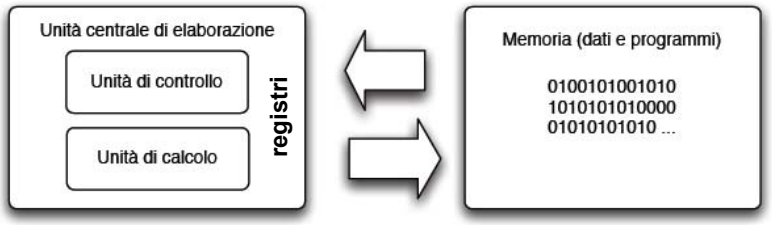


Figura 2.5: *La macchina di von Neumann*

Il programma è parte mutevole e risiede nella memoria.
Le istruzioni quindi possono mutare nel tempo e il calcolatore è riprogrammabile – Macchina Universale

La macchina di von Neumann (2)

- L'elaborazione si svolge come segue:
  - un'istruzione e i dati che tale istruzione deve manipolare vengono trasferiti o "caricati" dalla memoria nei registri
  - l'unità centrale esegue l'istruzione
  - gli eventuali risultati vengono trasferiti dai registri alla memoria
  - si passa all'istruzione successiva (o a quella specificata dall'istruzione di controllo, se l'istruzione eseguita era di questo tipo).
- L'elaborazione è sequenziale: viene eseguita un'istruzione per volta

La macchina di von Neumann (3)

- Un programma per una macchina di Von Neumann consiste, pertanto:
  - nella descrizione/prescrizione di una sequenza di istruzioni elementari sui dati
  - combinate con istruzioni di controllo, che modificano l'ordine nel quale eseguire le operazioni sui dati
- A differenza di altre precedenti computer la macchina di von Neumann mette i dati e i programmi allo stesso piano rendendo l'esecutore – computer – riprogrammabile N volte
- La Pascalina invece era progettata solo per le addizioni

Algoritmi ed efficienza

- Per risolvere un problema possiamo avere diversi algoritmi e tutti risolutori e validi
- Come scegliere l'algoritmo migliore?
  - Correttezza, comprensione ed eleganza, efficienza
- Se l'esecutore è un calcolatore universale cercheremo di scrivere algoritmi **efficienti** ossia che al crescere dei dati in input il numero delle operazioni svolte sia il migliore
- Lo studio dell'efficienza degli algoritmi porta a studiare la complessità computazionale degli algoritmi stessi secondo due parametri:
  - Tempo – quanti giri di CPU compie il computer
  - Spazio – quanta memoria occupa

Complessità computazionale

- La branca della teoria della complessità che studia quanto tempo e quanta memoria occorre per eseguire un algoritmo per portare a termine la risoluzione del problema
- Si valuta un ordine di grandezza in base al crescere dei dati inseriti
- Non è possibile un calcolo esatto quindi si esegue una stima sulla base del caso migliore e del caso peggiore

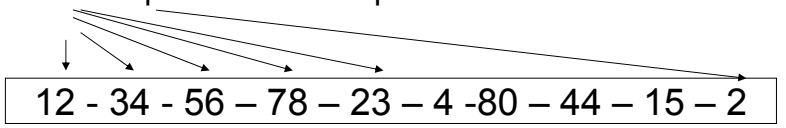
Esempio della ricerca sequenziale

- Ricerca di un elemento in un insieme ordinato di elementi per esempio un numero telefonico in un elenco ordinato alfabeticamente o un numero intero in un insieme ordinato di numeri interi
- Siano dati 100 numeri interi; la domanda è: il numero N (per es. 45) è contenuto nell'insieme?
- Algoritmo:

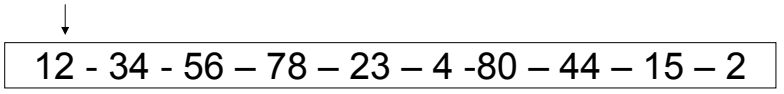
Tre variabili: *N; Elemento dell'insieme, Risultato*
Acquisisci *N* e *Insieme di elementi*
Imposta *Risultato* a `Insuccesso`
Ripeti la scansione degli elementi finchè *Risultato* vale `Successo`
OR fino alla fine dell'insieme
Confronto l' elemento successivo dell'insieme con N
Se sono uguali allora imposta *Risultato* a `Successo`
Comunica la mondo il valore di *Risultato*

Esempio della ricerca sequenziale

- N=2 - 10 passi necessari per trovare il numero 15



- Caso peggiore
- N= 12 – un passo per raggiungere il numero 12



- Caso migliore
- Caso medio (1+10)/2

Ricerca sequenziale

- Se gli elementi dell'insieme sono 100
- Caso migliore: 1
- Caso peggiore: 100
- Caso medio: (se N è nella prima posizione il numero dei tentativi è 1) + (se N è nella seconda posizione il numero dei tentativi è 2) + ecc..
  - Allora il caso medio è (1 + 2 + 3 + …+ 100) / 100
  - Più in generale (1 + 2 + 3 + …+ N) / N
  - Vale N * (N+1)/2
  - Diviso N dà (N+1)/2 che se N è 100 vale 50
- In sintesi (100+1)/2 = 50

Ricerca binaria

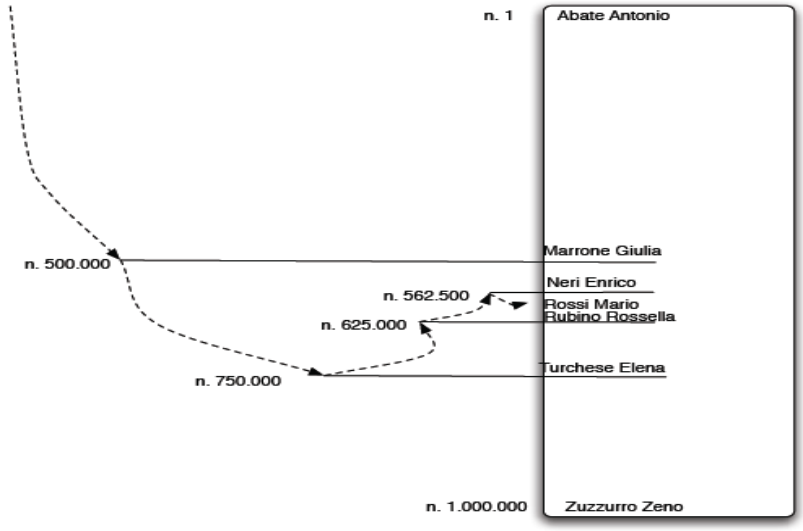


Figura 3.8: *La ricerca binaria*

Ricerca binaria

- Si suppone di avere una lista ordinata e di cercare Rossi Mario
- Si suppone di dividere la lista in due e vedere se il nome cercato cade nella prima parte della lista o nella seconda
- Si identifica così la parte su cui lavorare e si procede ricorsivamente ossia applicando lo stesso procedimento al risultato del passo precedente (pag. 85 Sartor)
- In questo modo si scarta tutta la parte di indirizzi che sicuramente sono inutili ai nostri fini e ci si concentra sulla parte utile

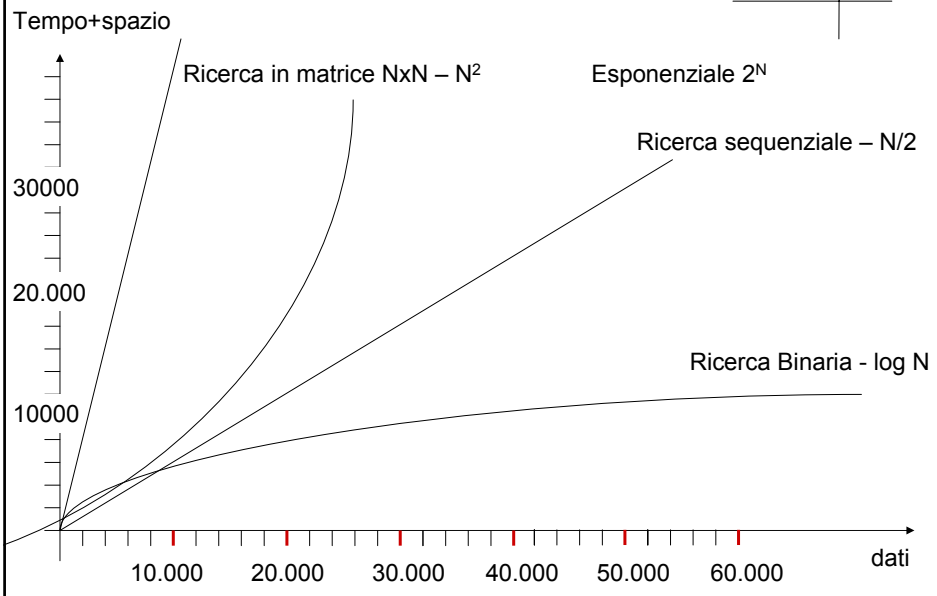
Ricerca binaria

- Si riduce il problema della metà (circa) a ogni interazione limitando la ricerca alla parte precedente o alla parte successiva del valore di metà dell'insieme
- Ogni tentativo permette di ridurre l'ampiezza del problema di un fattore 2
- Il primo tentativo riduce il numero delle possibilità a *N/2*; il secondo a *N/4*, ecc.
- Nel caso peggiore si dovrà proseguire finché rimane una sola possibilità
- Il numero di passi nel caso peggiore è *N/2/2/2/…/2 = 1*
- Il numero di passi nel caso migliore è N/2
- In generale avremo k divisioni per 2; dove k è il numero dei tentativi necessari a trovare il valore cercato
- Caso medio di passi è $N/2^k$ dove K è il numero di passi quindi $K = log_2 N$

Misure nel caso medio

Valore di N	***Ricerca lineare N/2***	***Ricerca binaria $Log_2 N$***
10	5	~4
100	50	~7
1 000	500	~10
10 000	5.000	~ 14
100 000	50.000	~ 17
1 000 000	500.000	~ 20

Grafico approssimato



Alcuni esempi

Quando *N* raddoppia il tempo di esecuzione …	**Si dice complessità …**
Non cambia – leggi il primo numero di una lista	Costante - N
Aumenta di una costante “piccola” – ricerca binaria	Logaritmica log N
Raddoppia – prodotto di due liste di numeri N	Lineare 2N
Un pò meno del raddoppio	N log N
Aumenta di un fattore 4 – ricerca in una matrice NxN	Quadratica N^2
Aumenta di un fattore esponente di 2 – sequenza di operazioni su una lista	polinomiale
Aumenta di molto secondo la base di cui N è esponente – torre di Hanoi	Esponenziale 2^N

Classi di complessità

Nome della classe	Espressione
Costante	Costante - N
Sottolineare	log N √N (radice di N)
Lineare	N o c*N dove c è una costante
Quadratica	N^2
Polinomiale	n^k +c*n
Esponenziale	2^N

Misure di efficienza

I problemi che si cerca di risolvere in modo automatico sono di tre categorie:

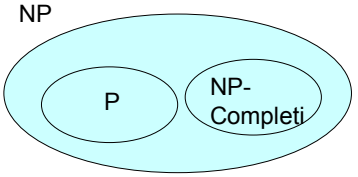
- Problemi per i quali sono al momento noti un certo numero di algoritmi risolutivi
- Problemi per i quali è noto che non esistono algoritmi risolutivi
- Problemi per i quali non si sa se esistano o non esistano algoritmi risolutivi

Kurt Gödel – 1906 - 1978

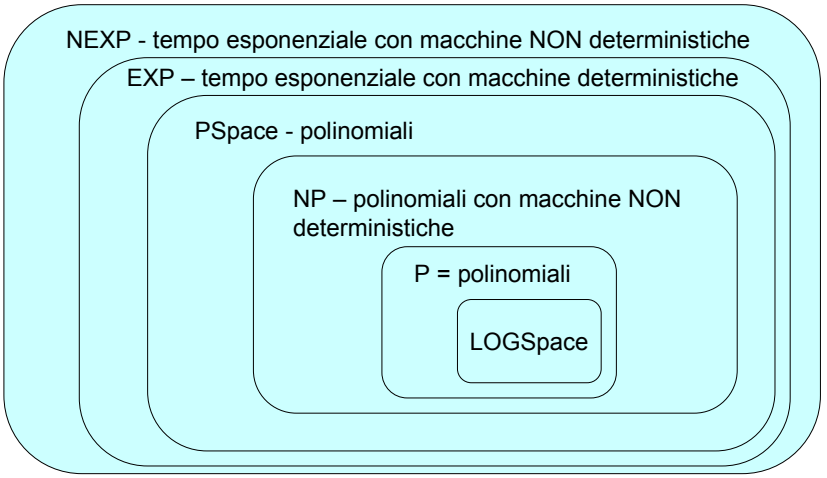
- Primo teorema di incompletezza di Gödel
  - Ogni teoria matematica coerente comprende sempre una formula è non dimostrabile e così pure anche la sua negazione
- Secondo teorema di incompletezza di Gödel
  - Nessun sistema coerente è sufficiente per dimostrare la sua coerenza
- Esistono problemi indecidibili di cui non si può dimostrare né che è vero né che è falso
- Esistono problemi irriducibili ossia irrisolvibili, altri ancora non sono calcolabili mediante tempi polinomiali, altri sono calcolabili in un tempo polinomiale

Classi di problemi

- I problemi quindi si possono dividere in tre grandi categorie:
  - Semplici – algoritmo polinomiale - P
  - Difficili – algoritmo non polinomiale (NP-completi)
  - Indecidibili – non si da dire nulla
- Le ultime due categorie sono problemi intrattabili



Classi di problemi



Materiali di riferimento e Domande possibili

- Capitolo 3 Sartor
- http://digilander.libero.it/unno2/sort/complessita.htm
- Differenza fra la macchina di Von Neumann e gli altri “computer”
- Definizione di complessità computazionale
- Efficienza in termini di tempo e spazio: cosa vuol dire?
- Caso peggiore, caso migliore, caso medio nella stima dell’efficienza di un algoritmo
- Le classi di complessità costante, lineare, quadatica, logaritmica, esponenziale.